

NAVAL POSTGRADUATE SCHOOL MONTEREY, CALIFORNIA



THESIS

**DYNAMICALLY DETERMINING DISTRIBUTION
STATISTICS FOR RESOURCES IN A DISTRIBUTED
ENVIRONMENT**

by

Thomas S. Cook

December 1999

Thesis Advisor:
Second Reader:

Taylor Kidd
Deborah Kern

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE December 1999	3. REPORT TYPE AND DATES COVERED Master's Thesis		
4. TITLE AND SUBTITLE: DYNAMICALLY DETERMINING DISTRIBUTION STATISTICS IN A DISTRIBUTED ENVIRONMENT		5. FUNDING NUMBERS		
6. AUTHOR(S) Cook, Thomas S.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT: Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE:		
13. ABSTRACT (maximum 200 words) Currently, the Department of Defense runs its special purpose applications on dedicated hardware (i.e., on "stovepipe systems"). Such hardware has inherent disadvantages. They have an inability to handle the resource contention that often occurs upon the influx of a large number of applications. A new application needing to use a given resource must typically wait for any preceding applications to first finish their use instead of searching out another capable resource. An even worse scenario is when the system fails and no applications can run until the system is repaired and brought back on-line. In all the cases, important decisions can potentially be delayed or made without important information. The Management System for Heterogeneous Networks (MSHN) will mitigate these deficiencies. The goal of MSHN is to manage several different types of applications across a changing heterogeneous network. MSHN determines the best resource on which to run an application based on both the applications and overall system's Quality of Service (QoS). The focus of this thesis is to write and demonstrate for MSHN the worth of an algorithm that can determine and update distribution statistics for the end-to-end QoS resource usage of an application program. These distributions are vital in assisting MSHN in the scheduling and rescheduling of applications across a network.				
14. SUBJECT TERMS Resource Management System, Distributed Systems, Client Library, Resource Monitoring, Stochastic Algorithms, Distribution, Heterogeneous Computing			15. NUMBER OF PAGES 141	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18 298-102

Approved for public release; distribution is unlimited.

**DYNAMICALLY DETERMINING DISTRIBUTION STATISTICS FOR
RESOURCES IN A DISTRIBUTED ENVIRONMENT**

Thomas S. Cook
Major, United States Army
B.S., Brockport State University, New York, 1987

Submitted in partial fulfillment of the
requirements for the degree of

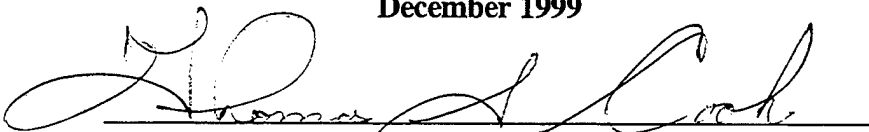
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

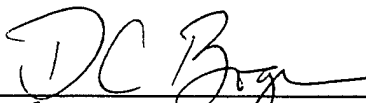
December 1999

Author:



Thomas S. Cook

Approved by:


for Taylor Kidd, Thesis Advisor

Deborah Kern, Second Reader



Dan Boger, Acting Chairman
Computer Science Department

ABSTRACT

Currently, the Department of Defense runs its special purpose applications on dedicated hardware (i.e., on "stovepipe systems"). Such hardware has inherent disadvantages. They have an inability to handle the resource contention that often occurs upon the influx of a large number of applications. A new application needing to use a given resource must typically wait for any preceding applications to first finish their use instead of searching out another capable resource. An even worse scenario is when the system fails and no applications can run until the system is repaired and brought back on-line. In all the cases, important decisions can potentially be delayed or made without important information. The Management System for Heterogeneous Networks (MSHN) will mitigate these deficiencies.

The goal of MSHN is to manage several different types of applications across a changing heterogeneous network. MSHN determines the best resource on which to run an application based on both the applications and overall system's Quality of Service (QoS). The focus of this thesis is to write and demonstrate for MSHN the worth of an algorithm that can determine and update distribution statistics for the end-to-end QoS resource usage of an application program. These distributions are vital in assisting MSHN in the scheduling and rescheduling of applications across a network.

TABLE OF CONTENTS

I. INTRODUCTION.....	1
A. BACKGROUND.....	1
B. SCOPE OF THIS THESIS.....	4
C. MAJOR CONTRIBUTIONS OF THIS THESIS.....	5
D. ORGANIZATION.....	5
II. MSHN.....	7
A. THE EVOLUTION OF MSHN.....	7
B. MSHN'S GOALS	12
C. MSHN COMPONENTS AND FUNCTIONALITY	14
D. THE NEED FOR DISTRIBUTION STATISTICS IN MSHN	17
III. OVERVIEW OF PREVIOUS WORK.....	23
A. RESOURCE MONITORING.....	23
1. Network Weather Service (NWS)	25
2. SmartNet	27
3. DeSiDeRaTa	28
4. Condor	30
5. Odyssey.....	31
6. MSHN.....	33
B. STOCHASTIC SCHEDULING	33
C. SUMMARY	34
IV. FORMAL DEFINITION OF THE PROBLEM	35
A. PROBLEM.....	35
B. APPROACHES TO SOLVING THE PROBLEM.....	35
1. Kolmogorov-Smirnov Test (K-S test)	35
2. Anderson-Darling Test (A-D test)	41
3. Non-Parametric Bayesian Approach (Dirichlet process).....	43
C. SUMMARY	44
V. MONITORING RESOURCES USING MSHN'SCLIENT LIBRARY.....	45
A. THE CLIENT LIBRARY	45
B. A WRAPPED APPLICATION	45
C. MONITORING RESOURCES	47
D. CLIENT LIBRARY OUTPUT.....	49
E. SUMMARY	53
VI. EXPERIMENT.....	55
A. DESIGN OF THE EXPERIMENT.....	55
1. Wrapping and running the application and pre-processing CL output.....	55
2. Designing and implementing the statistics application.....	57
B. EXPERIMENT METHODOLOGY	59
C. RESULTS	60
D. CONCLUSIONS.....	65

E. SUMMARY	66
VII. SUMMARY AND FUTURE WORK.....	67
A. SUMMARY	67
B. FUTURE WORK	68
APPENDIX A: CODE FOR DATA OBJECT AND STATISTICAL TESTS	71
DATAOBJECT	71
MYFILEIOCLASS	78
MYSTATSPROGRAM.....	83
STATTEST	84
STDNORMALTABLE	99
APPENDIX B: MSHN WRAPPER OUTPUT: BISON APPLICATION	103
READ LOCAL FILE BETWEEN CALL INTERVAL DATA.....	103
READ LOCAL FILE SYSTEM CALL INTERVAL DATA	105
WRITE LOCAL FILE BETWEEN CALL INTERVAL DATA	107
WRITE LOCAL FILE SYSTEM CALL INTERVAL DATA.....	109
WRITE REMOTE FILE BETWEEN CALL INTERVAL DATA	111
WRITE REMOTE FILE SYSTEM CALL INTERVAL DATA	112
APPENDIX C: EXPERIMENT RESULTS	113
READ LOCAL FILE BETWEEN CALL INTERVAL RESULTS.....	113
READ LOCAL FILE SYSTEM CALL INTERVAL RESULTS	113
WRITE LOCAL FILE BETWEEN CALL INTERVAL RESULTS	114
WRITE LOCAL FILE SYSTEM CALL INTERVAL RESULTS.....	115
WRITE REMOTE FILE BETWEEN CALL INTERVAL RESULTS.....	115
WRITE REMOTE FILE SYSTEM CALL INTERVAL RESULTS	116
APPENDIX D: MINITAB RESULTS.....	117
APPENDIX E: ACRONYMS.....	123
LIST OF REFERENCES	125
INITIAL DISTRIBUTION LIST	127

LIST OF FIGURES

Figure 1: MSHN Architecture with Permission From [HENS99]	3
Figure 2: Schedule Comparison with Permission From [KIDD99].....	9
Figure 3: Computer Heterogeneity with Permission From [KIDD96]	10
Figure 4: Partitioned Runtime with Permission From [KIDD96].....	11
Figure 5: Physical Instantiation of MSHN Architecture with Permission From [HENS99].....	14
Figure 6: Scheduling Algorithms Using only the Estimated Mean	18
Figure 7: Job Runtime Distributions	20
Figure 8: SmartNet Architecture From [KIDD96].....	27
Figure 9: Resource Information Flow of DeSiDeRaTa	29
Figure 10: The Condor Pool From [SCHN98].....	30
Figure 11: Odyssey Client Architecture After [NOBL97].....	32
Figure 12: Hypothesized Distribution vs. Observed Distribution After [LAWK91].....	36
Figure 13: cdf Convergence Using the Dirichlet Process	44
Figure 14: Ordinary Linking of Bison vs. Wrapped Linking of Bison.....	46
Figure 15: Event Flow Diagram for read () From [SCHN98].....	48
Figure 16: Some Resource Requirement Information Output by Initial Prototype of MSHN CL	49
Figure 17: Example Output From the MSHN Client Wrapper (in sec)	51
Figure 18: Example Execution Timeline of a Bison Application	52
Figure 19: Example of Data Elements Extracted From MSHN CL Using Perlscript.....	57
Figure 20: Descriptive Statistics and Histogram of WRFBCI Data Computed using Minitab	63
Figure 21: Minitab Results For rlfbcf Data.....	117
Figure 22: Minitab Results For rlfsci Data	118
Figure 23: Minitab Results For rlfsci Data	119
Figure 24: Minitab Results For wlfsci Data.....	120
Figure 25: Minitab Results For wrfbci Data	121
Figure 26: Minitab Results For wrfsci Data.....	122

LIST OF TABLES

Table 1: Job Execution Lengths with Permission From [KIDD96].....	8
Table 2: Sample Data from a Normal Population.....	37
Table 3: Results of $F(X_{(i)})$ $i = 1$ to 10.....	39
Table 4: Modified Critical Values for Adjusted K-S Test Statistics From [LAWK91]	40
Table 5: Modified Critical Values for Adjusted A-D Test Statistics From [LAWK91].....	42
Table 6 : "Write Remote File Between Call Interval" Data Captured using the MSHN Client Library	61
Table 7: Descriptive Statistics for WRFBCI Data	61
Table 8: " Goodness Of Fit " Test Results using Our "Goodness Of Fit" Application	62

ACKNOWLEDGEMENTS

I would like to sincerely thank my best friend and wife Denise and my son Tommy for their unwavering support, love, and understanding. Thanks to Mom and Dad for absolutely everything. You are my drive. To the brothers Cook (John, Bill and Steve) and their fabulous families, thanks for your support. Thank you to Dr. Taylor Kidd, CDR. Deborah Kern, and Dr. Lyn Whitaker for their insight, expertise, and extreme patience. Finally, thank you to two of the greatest friends I have ever met, LT. Kurt Rothenhaus and LCDR. Wayne Porter, you guys helped me make it through this school and made every day a barrel of laughs.

I. INTRODUCTION

This thesis investigates some of the issues and problems associated with acquiring distribution statistics for an application's resource usage while running in a heterogeneous distributed network. These distribution statistics are an aggregated and more usable form of the raw data provided by the Client Library in the Management System for Heterogeneous Networks (MSHN). The resulting distributions from this study can be used by MSHN to determine, for all machines on the network and with some probability, an application's runtime, required memory, and data stream statistics, among many others. MSHN can also use this information to help in the scheduling and rescheduling of applications across the network.

A. BACKGROUND

In the early 1990's, the Heterogeneous Computing Team at the US Navy's NCCOSC RDTE Division in San Diego developed a scheduling framework called SmartNet for managing jobs and resources in a heterogeneous computing environment. The designers of SmartNet incorporated six innovative techniques to improve its performance over other types of Resource Management Systems (RMSs). The six innovations were (1) how SmartNet recognized and exploited heterogeneity, (2) its development of Compute Characteristics, (3) its ability to handle uncertainty, (4) how it accounted for the sharing of resources in a distributed environment, (5) its view of optimization criteria(s), and (6) the methods employed by it to search the scheduling space [KIDD96]. In addition, SmartNet has the ability to make the scheduler aware of two very important elements in a heterogeneous network that previous distributed environments did not: (1) the load on the machines in the network, and (2) an application's affinity for a particular machine on the network. During its heyday, SmartNet ran in several computing centers across the United States. For a full description of SmartNet, see [KIDD96]. SmartNet has spawned other projects in the research area of distributed computing in heterogeneous environments. One such project is the Management System for Heterogeneous Networks (MSHN).

MSHN is a DARPA sponsored project funded to investigate and demonstrate, in a Department of Defense (DoD) context, an environment that supports and illustrates the usefulness of adaptive applications. Currently, the DoD runs its special purpose applications on dedicated hardware (i.e., on "stovepipe" systems). An inherent disadvantage in using a stovepipe system is its inability to handle the resource contention that often occurs upon the influx of a large number of tasks. In such systems, a new application needing to use a given resource must typically wait for any preceding applications to first finish their use instead of searching out another capable resource. An even worse scenario occurs when the system fails and no applications can run until the system is repaired and brought back on line. In either case, important decisions can potentially be delayed or made without important information. MSHN will mitigate these deficiencies. The goal of MSHN is to manage several different types of tasks across a changing heterogeneous network. MSHN determines the best resource on which to run an application based on both the application and overall system's Quality of Service (QoS). Some factors influencing QoS include security, deadlines, priorities, adaptability and resource availability. Much of this information is stored in two databases/status servers in the MSHN architecture (see Figure 1 below).

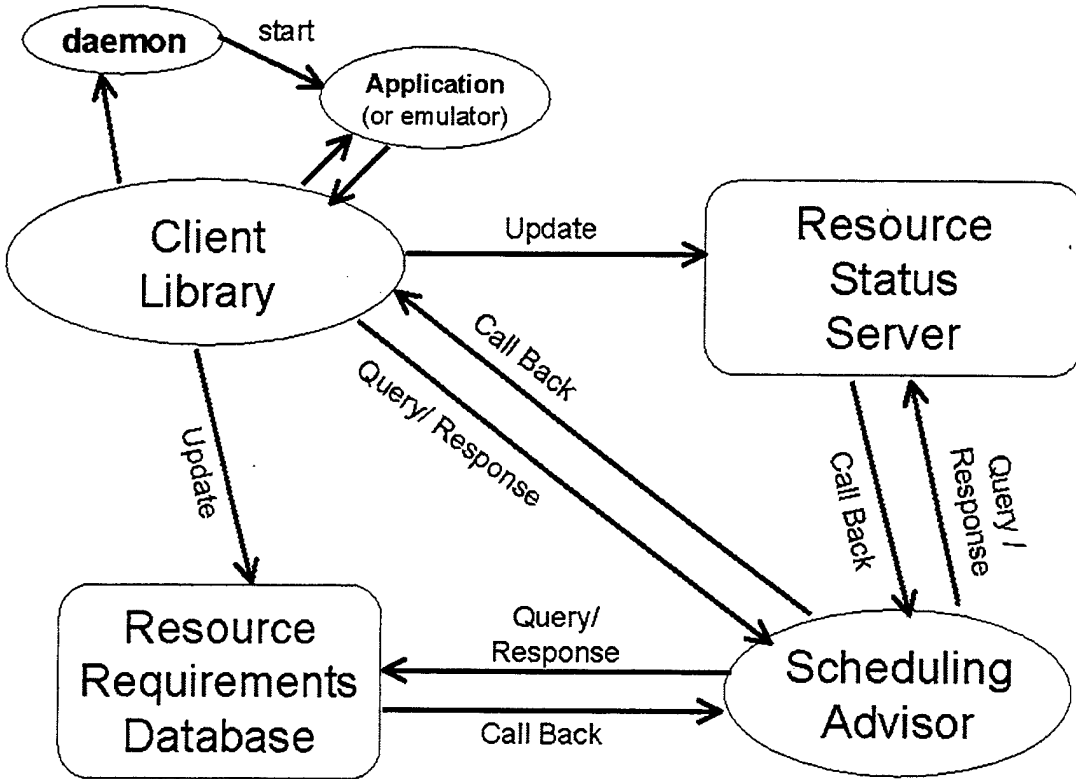


Figure 1: MSHN Architecture with Permission From [HENS99]

To date among many other things, MSHN has demonstrated the ability to collect elemental QoS data (e.g., CPU time, total memory used, and network latency) from adaptive applications [SCHN99, PORT99]. The focus of this thesis is to develop and demonstrate the worth of algorithms that determine and update the distribution statistics of this elemental data. For example, the resulting distributions from this study can be used by MSHN to better determine, for all machines on the network and with some probability, an application's system and user runtimes, and required memory. MSHN can also use this

information to help determine the scheduling and rescheduling of applications across the network.

B. SCOPE OF THIS THESIS

The MSHN architecture consists of ten major components: (1) the Scheduling Advisor, (2) the Client Library, (3) the Resource Status Server, (4) the Resource Requirements Database, (5) the MSHN Daemon, (6) an application, (7) the Application Emulator, (8) an adaptation-aware application, (9) resources, and (10) the Visualizer. The functionality of these components are discussed in Chapter II. This thesis focuses primarily on the Client Library (CL), Resource Requirements Database (RRD), and Resource Status Server (RSS). The CL serves many purposes. It links with legacy, adaptive, and MSHN-aware applications providing (1) a transparent interface to all of the other components, and (2) a mechanism for intercepting system calls for the collection of resource usage and status information. The collected information is then forwarded to and stored in the RSS and the RRD for use by the Scheduling Advisor (SA) and the Visualizer (VIS).

The first part of the objective of this thesis is to design an algorithm that calculates the distribution statistics for both the resource usage of applications linked with the CL, and the resource status of the various resources on the network (e.g., computers and the network itself). The distribution statistics can then be stored in the RRD and the RSS for use by the SA and VIS.

Specifically, this thesis answers the following questions:

- Can specialized algorithms, ones that rely on knowing the underlying distribution of the data, be useful in MSHN or is a more generalized stochastic algorithm needed?
- What parameters or set of parameters should be stored in the RSS and RRD? How much of the data produced from the client wrapper should be maintained for use in determining the distribution statistics?

C. MAJOR CONTRIBUTIONS OF THIS THESIS

This thesis makes contributions to MSHN and the DoD. First, this thesis provides procedures for aggregating and analyzing data collected by MSHN's Client Library. These procedures aggregate raw data in to a statistical form more usable for decision making by the MSHN components. Secondly, this thesis demonstrates that these statistics are of interest.

This research benefits the DoD in supporting application mixes which will be sharing resources in the future. As the DoD moves away from "stovepipe" systems and towards commercial-off-the-shelf (COTS) systems, we must ensure that those systems (1) can share resources (dedicated hardware will no longer be practical for most real-time applications), and (2) in a crisis situation, can best manage the use of and maximize the benefits of those resources. This study aids in understanding how to best move towards such COTS systems.

The procedures and research in this thesis focused on aggregating and analyzing data for MSHN. However, it should be clear that this research and these procedures, with slight modifications, could be applied to numerous data aggregation and analysis problems.

D. ORGANIZATION

Each chapter of this thesis begins with an introduction and ends with a brief summary. The body of this thesis is organized as follows: Chapter II discusses the goal of MSHN, describes the functionality of the MSHN components, and explains where and how this research fits into the MSHN program. Chapter III examines resource monitoring systems, and the potential of future scheduling algorithms that can make use of statistical information. Chapter IV states the definition of the problem, enumerates and explains possible approaches, and discusses possible solutions. Chapter V describes how an application is wrapped using the MSHN Client Library. Chapter VI explains the design of the experiment, and discusses and provides an analysis of the results. The final chapter provides a summary of the work done in this thesis and presents possibilities for future work.

II. MSHN

The primary purpose of this chapter is to illustrate the importance of being able to calculate distribution statistics for MSHN. For this illustration to be clear, the reader must have some level of understanding of how MSHN evolved, what MSHN's goals are, and how the components of the MSHN architecture function. A tremendous amount of research has been conducted in the area of resource management and none more important than that conducted by the Heterogeneous Computing Team at the US Navy's facility at the NCCOSC RDTE Division in San Diego. This initial research eventually led to MSHN.

A. THE EVOLUTION OF MSHN

A heterogeneous computing environment executes many different types of I/O-intensive and/or compute-intensive applications on many different types of computers. In such an environment, the assignment of jobs to resources is generally done using one of two basic systems, a Resource Management System (RMS) or a Distributed Operating Systems (DistOS). The main goal of these types of systems is to transparently give simultaneous users direct access to many different types of powerful computers upon which to run their applications. When users schedule jobs on local hosts, two important elements are taken into consideration: (1) how busy the machine is (the *Load*), and (2) how well a job runs on that particular machine (the *Affinity*). For example, if other capable machines are available, scheduling a job on a machine that is currently running several other applications may not make sense. On the other hand, scheduling a job on a machine just because no other applications are running on it may not make sense either, especially if the job does not execute well on that particular machine. Generally speaking, the two systems mentioned above only consider machine load in their scheduling policies. A RMS client runs on a host while accepting jobs, and then schedules those jobs on machines that have the lightest load. In contrast, a distributed operating system controls and manages its hardware and software resources in a manner such that its users view the entire system as a powerful monolithic computer system.

The example presented in [KIDD96] compares the times at which the last job completes of three schedulers using three different scheduling policies. The scenario has

three machines, A, B and C, and four jobs, 1, 2, 3 and 4. Each scheduler computes a schedule based on its scheduling policy. The three scheduling policies used consider load, affinity, and a combination of load and affinity. Scheduler 1 used Opportunistic Load Balancing (OLB) as its scheduling policy. OLB is similar to the policies used by most RMSs and DistOSs in that it assigns the next queued job to the next available machine. Scheduler 2 used a policy called Limited Best Assignment (LBA). LBA assigns each job to the machine where that job is expected to run the fastest, assuming that all machines are empty. Scheduler 3 assigns jobs according to both their affinity for a machine and the load on that machine. To keep the experiment simple, the authors of [KIDD96] made the following three assumptions: (1) every job executes for exactly the predicted amount of time, (2) jobs all arrive simultaneously, and (3) jobs were queued in the specific order as seen in Table 1 below.

JOBS	MACHINES		
	A	B	C
1	4	17	7
2	5	11	6
3	4	16	8
4	11	4	9

Table 1: Job Execution Lengths with Permission From [KIDD96]

Using the job execution lengths given in Table 1, the three schedulers produce the following schedules shown below in Figure 2.

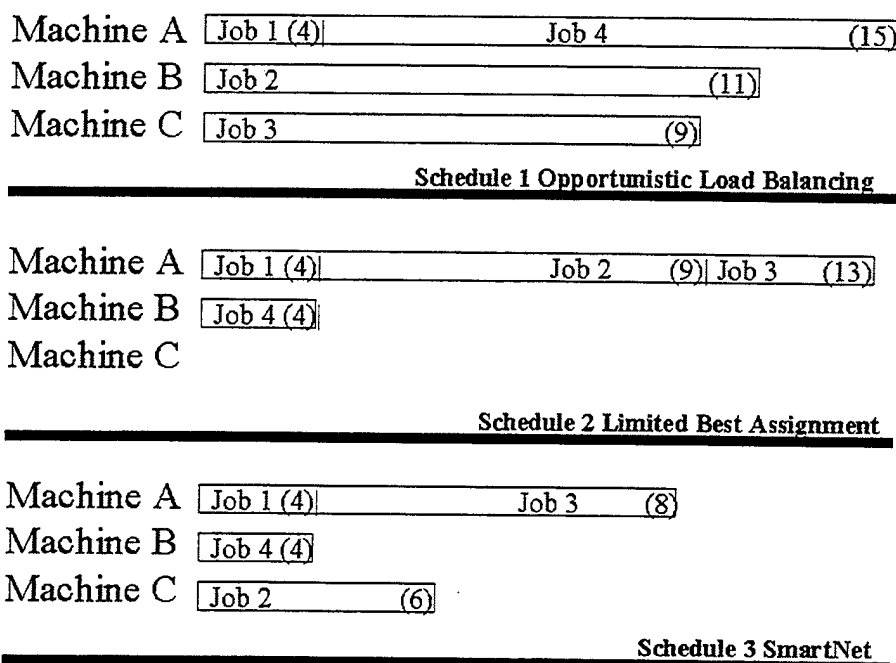


Figure 2: Schedule Comparison with Permission From [KIDD99]

From Figure 2, it is clear that the schedulers whose policies only considered one of the elements, either load or affinity, performed significantly slower than the SmartNet scheduler, which considered both elements. In fact, the SmartNet scheduler completed the jobs in roughly half the time of the other two schedulers. With few exceptions, policies used in most RMSs and DistOSs only consider the load on a particular machine when scheduling jobs, much like Scheduler 1. From Figure 2, we can also see that in a heterogeneous computing environment better schedules are possible when a scheduler considers both how busy machines are and how well applications run on each of the machines. A scheduling policy considering both load and affinity is used in SmartNet. This policy was one of several innovations that improved SmartNet's performance over that of other existing RMSs. The other SmartNet performance innovations included its

ability to effectively deal with heterogeneity, the development of compute characteristics, its ability to deal with uncertainty, how it accounted for resource sharing in a distributed environment, its view of optimization criteria(s), and its methods for searching the schedule space [KIDD96].

SmartNet was designed to take advantage of the computer heterogeneity inherent in a distributed networked environment. As discussed above, certain jobs have an affinity for certain machines. One of SmartNet's objectives was to find the machine upon which a job would execute the fastest. Figure 3, taken from [KIDD96], shows how performance is maximized when the RMS can match a job's type to the machine upon which the job executes best.

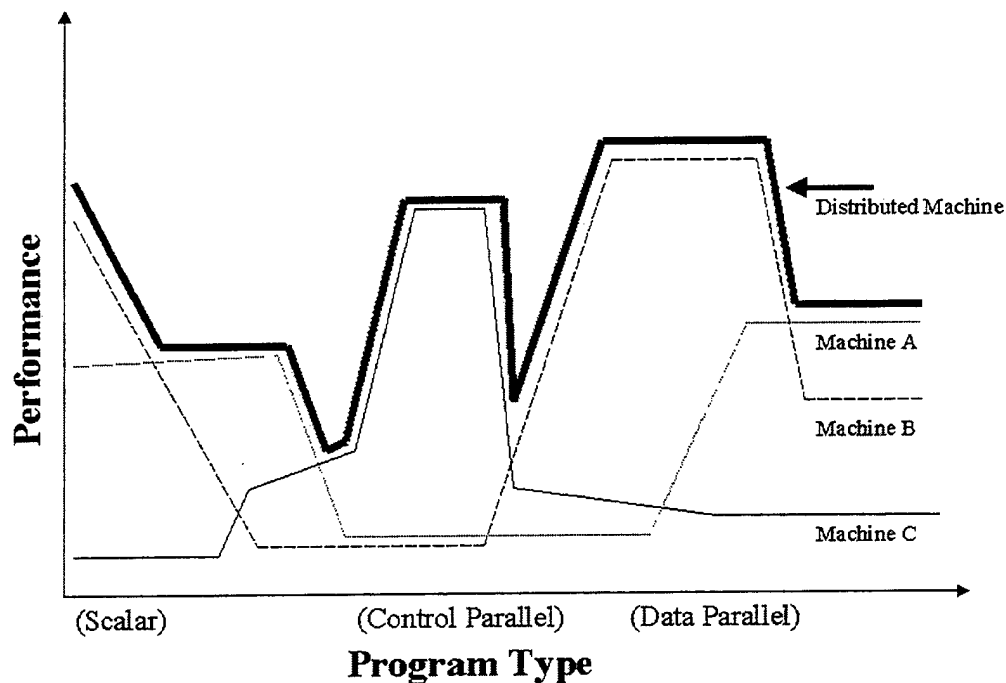


Figure 3: Computer Heterogeneity with Permission From [KIDD96]

SmartNet's ability to produce quality schedules relied heavily upon its being able to predict good estimates of job runtimes. If SmartNet knew the exact runtimes of jobs a priori, scheduling would be much less difficult to perform. Unfortunately this was not the case. The next best thing to having exact job runtimes is having the runtime distributions of those jobs, provided the distribution is uni-modal and has a narrow variance. Again, this is typically not the case. In fact, job runtime distributions are often just the opposite in that they have a wide variance and are generally multi-modal. To deal with these multi-modal distributions, the SmartNet Team partitioned the distribution into pieces. Compute Characteristics and Compute Characteristic Operating Points (CCOPs) define the partitions. Figure 4 below shows a typical runtime distribution and how it might look before and after being partitioned. This instance shows a job with a single Compute Characteristic and three different CCOPs.

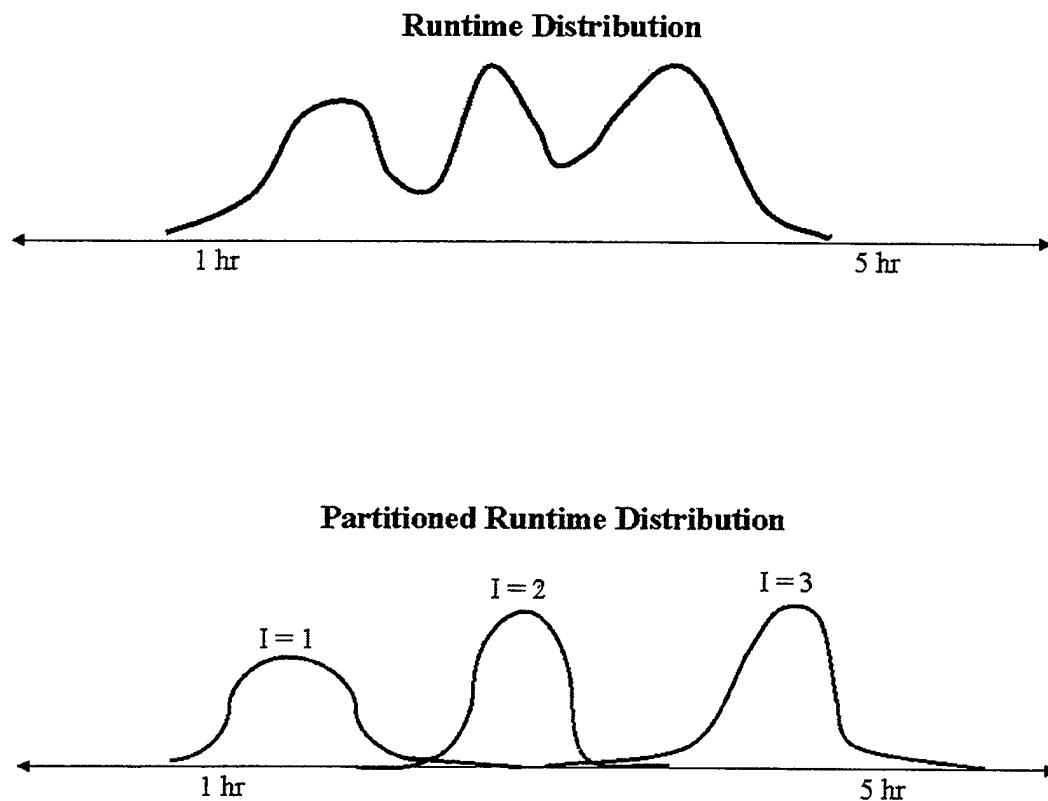


Figure 4: Partitioned Runtime with Permission From [KIDD96]

Distributed environments are non-deterministic in nature for reasons such as resources being shared and machines operating asynchronously. In order to improve on performance, SmartNet tracked and took into account the uncertainty caused by non-determinism.

The final innovation of SmartNet was that it included in its Scheduler separate optimization and search engines. The criterion in the optimization engine defined the best possible solution in the space the search engine explored.

All of the above innovations were vital to SmartNet's success. SmartNet has made significant contributions to several government agencies such as the DoD, the National Institute of Health, and NASA. Perhaps even more important was the research conducted to build SmartNet. Today, much of the research on computing in a distributed and heterogeneous environment is a continuation of or is incidental to SmartNet. MSHN is one of many projects that have benefited from the work done on the SmartNet project. For a much more detailed description of SmartNet's architecture and how SmartNet operated, see [KIDD96].

B. MSHN'S GOALS

An important difference between MSHN and SmartNet is that SmartNet was designed to be a fully functional and implemented product, while MSHN is a research system. MSHN is built and tuned through experimentation with the desire of determining the best ways to build a RMS, and has much broader goals than SmartNet. MSHN has three overarching goals. First, MSHN has to account for the overhead of jobs sharing resources. This has a significant impact on mapping and scheduling. Second, MSHN needs to support adaptive and adaptive-aware applications. Adaptive applications, as defined by the MSHN researchers, are idempotent applications that can exist in several different versions [HENS99]. Third, MSHN needs to provide good Quality of Service (QoS) to several different sets of simultaneous users, each of whom may be executing different types of jobs.

The application model used by MSHN is much more complex than that used in SmartNet. In SmartNet, applications first acquire data from a data repository, then compute results based on the data gathered, and finally write the results back to a possibly

different repository. Because acquiring the data and writing the data back to the repository are of significantly shorter duration than computing the results, SmartNet assumed there was no contention for either the network or the data repositories. MSHN's applications go through many more phases, each of variable length, and therefore MSHN must account for the resulting overhead. These phases are discussed in Section C of this chapter.

MSHN has the ability to perform a cost benefit analysis, terminating if necessary the current version of an application in favor of a version that will better meet the user's QoS expectations. For example, suppose a user has two applications, one that can generate and display a full video of the latest weather patterns, and another that instead produces a succession of still photos of the patterns. The user prefers the video to the photos but needs to see one or the other immediately. At the time the user submits this application, MSHN determines that there are insufficient resources to run the video (e.g., too little bandwidth) and the still photos are shown instead. If, after a short period of time, enough bandwidth becomes available, MSHN may decide to terminate the photo application and begin executing of the video application once again. Though an over simplification of the process, this scenario is an excellent example of an application adapting to a changing environment. It is important to note that an adaptive application does not necessarily adapt without some sort of user interaction, either at the time the application is submitted or during its execution.

Another application that MSHN can manage similar to the adaptive application is called an adaptive-aware application. An adaptive-aware application has the ability to sense changes in its environment. Therefore, an adaptive-aware application can sense the decline of a resource (e.g., bandwidth) and adapt. In the above scenario, perhaps the adaptive-aware application will show the still photos. In addition, the adaptive-aware application potentially has the ability to migrate, i.e., store its state when it terminates and restore its state upon execution. For example, if an adaptive-aware application detects a significant loss of bandwidth, it can take the following actions: stop its execution in place, store its current state, and use the saved state to immediately start presenting the still photos at approximately the same spot at which the video was terminated. [HENS99]

Finally, MSHN researchers are defining a measure that encapsulates many different QoS requirements. This measure is optimized and then used by MSHN to map

applications to resources. Much of how the above goals are going to be accomplished is presented in the following section on MSHN components and functionality.

C. MSHN COMPONENTS AND FUNCTIONALITY

This section is dedicated to describing how the separate components of MSHN communicate and function. It presents a general overview of the system as a whole followed by the functionality of the individual components.

As seen in Chapter I, Figure 1, MSHN has six main components. These components can all exist on the same machine or can be distributed throughout the network on different machines. The Daemon is the only component that must be present on all machines in the MSHN environment. A possible MSHN set-up is presented in Figure 5 below.

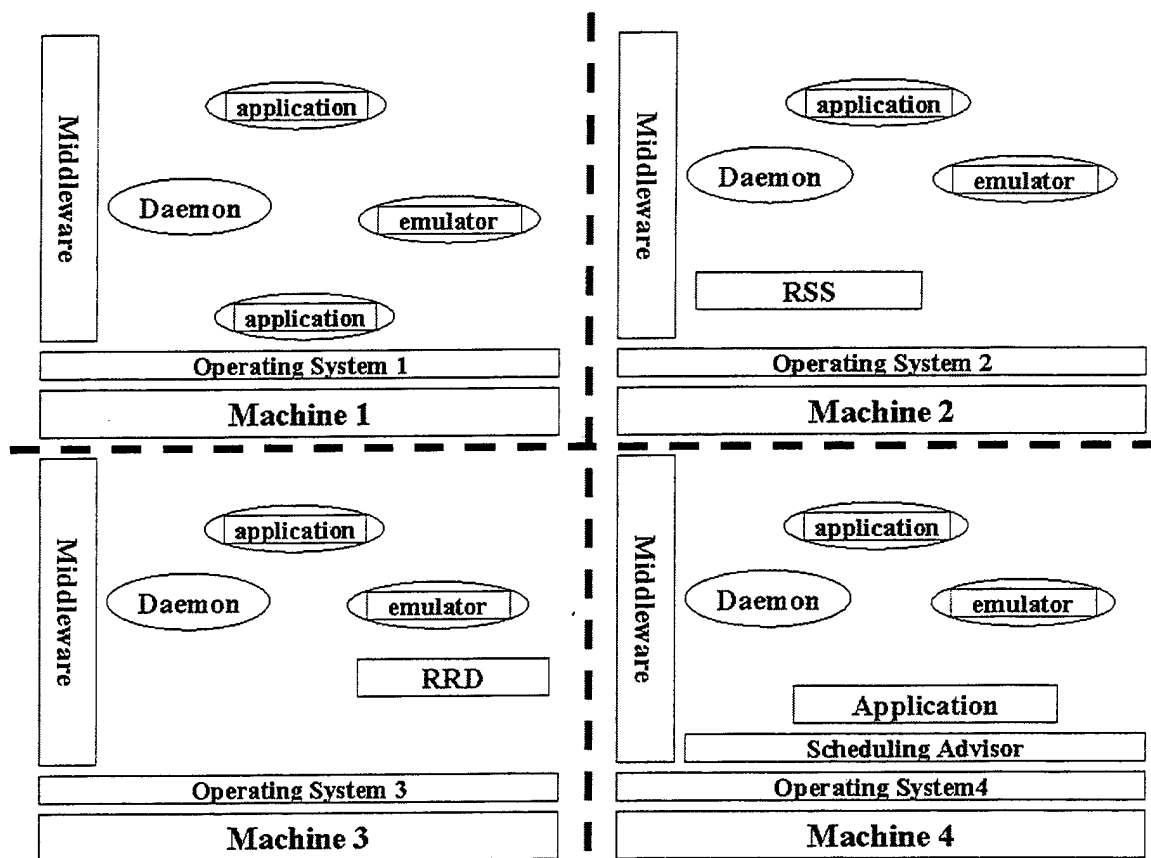


Figure 5: Physical Instantiation of MSHN Architecture with Permission From [HENS99]

When an adaptive or adaptive-aware application is submitted to MSHN for execution, it is "wrapped" in MSHN's Client Library (CL). The CL is the one MSHN component that communicates with all of the other MSHN components. The CL intercepts an application's calls to system libraries. Before executing a new process, the CL references a list of applications managed by MSHN. If the application is not on the list, MSHN passes the request to the local OS. If the application is on the list, the request is passed to the MSHN Scheduling Advisor (SA). The SA determines where to execute the application or process. This decision is based mostly on information queried from the MSHN Resource Requirements Database (RRD) and the MSHN Resource Status Server (RSS). The RRD is a database that stores information pertaining to and gathered on which resources and how much of those resources a particular application requires. The RSS is similar to the RRD in that it too is a database storing resource-related information. The RSS stores current status and availability information on all of the resources available to MSHN. Both the RRD and the RSS store data gathered via the CL. The CL passively monitors and reports an application's resource usage to the RRD and the current status of resources used by the application to the RSS. Based on the advice of the SA, the CL contacts the MSHN Daemon on the appropriate resource and requests that the process be started on that particular machine. That Daemon then starts executing the process.

The final component is the MSHN Application Emulator (AE). The AE's purpose is twofold. First, it mimics a real application without the overhead of re-coding, installing, maintaining, and running the real application. Its execution can help in initially populating the RRD. Second, the AE can be used to sense the status of resources upon which no MSHN applications are running, ensuring that the RSS is being populated with accurate data. What follows is a more detailed description of the MSHN components important to this thesis research and the purpose those components serve. The components are the CL, the RRD, and the RSS.

The Client Library, commonly referred to as "the wrapper," was designed and tested by Mathew Schnaidt [SCHN98]. It was later modified and implemented by the MSHN staff. As discussed above, the CL is linked to the application in order to intercept

system calls and report resource usage and resource status to the RRD and RSS, respectively. The following is a list of the resources that the prototype wrapper monitored:

- Total runtime
- Time blocked waiting for user input
- Local I/O
 - Total number of bytes read/written
 - Total number of reads/writes
- Network file I/O
 - Time to read from remote disk
 - Total number of bytes read/written
 - Total number of reads/writes
- Network I/O
 - Total number of bytes read/written
 - Estimate of latency seen by process
 - Estimate of throughput seen by process
- Local IPC
 - Total number of bytes read/written
 - Total number of reads/writes

In addition, the following resource information is available through system calls and utilities:

- Total memory used
- Number of page faults
- CPU time and user time

The above information is what was included in the CL in the initial prototype version. Since then, Shirley Kidd, a member of the MSHN staff, has modified the CL so that it now has the ability to collect the following fine grain information:

- User CPU time
- System CPU time
- Between system call time

Currently, the above information is forwarded to and stored in the RRD as raw data. For example, the CL may record the wall-clock time for application X as 3.34 seconds. The next time application X is executed, the CL may record the wall-clock time as 4.29 seconds. Each and every time application X is executed a separate wall-clock time is recorded and maintained in the RRD. The same is true for every other resource measurement that the CL collects. Each measurement is stored in the appropriate data repository (RRD or RSS).

The RSS actually maintains three types of information: short term, medium term and long term. This information comes from either the CL or a system administrator. When the SA makes a request concerning a particular resource to the RSS, the RSS returns its most recent estimate of the resource's current availability. The SA establishes callbacks to the RSS if the resource availability thresholds are surpassed or if a CL update frequency requirement needs to be met.

The design of the RRD is similar to that of the RSS. The RRD contains information about resource usage of applications. This information is passed to the SA. The RRD establishes callbacks to the SA when thresholds are surpassed and to meet any update frequency requirements. The RRD receives its updates from the CL.

The above are the components central to this thesis. For a detailed description of all MSHN components, how MSHN was built, how MSHN functions, and a tutorial on how to wrap an application, see Matt Schnaidt's thesis [SCHN98].

D. THE NEED FOR DISTRIBUTION STATISTICS IN MSHN

The case for why distribution statistics are needed in MSHN comes from a recent paper written by Dr.'s Taylor Kidd and Debra Hensgen of the Naval Postgraduate School. They show that scheduling algorithms that use both the expected runtime and their distributions can provide better schedules than those algorithms that rely solely on the expected runtime (see [KIDD99].) The paper focuses primarily on the performance of the load-and-affinity policy, discussed in Chapter I, when the actual runtimes of jobs can differ from the expected runtimes. The fundamental problem with most RMS scheduling algorithms is that they assume resource usage is deterministic. This is a poor assumption. While it is possible for the estimated usage of a resource to equal actual usage, it is not

probable. Through experimentation and common sense, we know that 50% of the actual usage is less than the expected usage and 50% is more. Figure 6 shows how the more sophisticated scheduling algorithms currently in use work, such as that used in SmartNet. Part A of Figure 6 shows six different jobs run several times on three different machines. The resource usage statistic we are focusing on in this example is job runtime. Part B of Figure 6 shows the average time (e.g., \bar{x}_{11}) of the recorded runtimes (e.g., $\bar{x}_{111}, \dots, \bar{x}_{11n}$) of the jobs on the three machines. The job is then scheduled to run on the machine upon which the job had the lowest expected runtime (e.g., Job 1 runs fastest on Machine 1). This schedule is illustrated in part C of Figure 6. Once the schedule is built, the predicted time at which the last job completes can be calculated. In our case, this predicted time is 11 time units. If each of the jobs in the schedule run for exactly their estimated time, the schedule works well. If any of the jobs in the schedule happen to run for more than their expected times, problems with the schedule can occur.

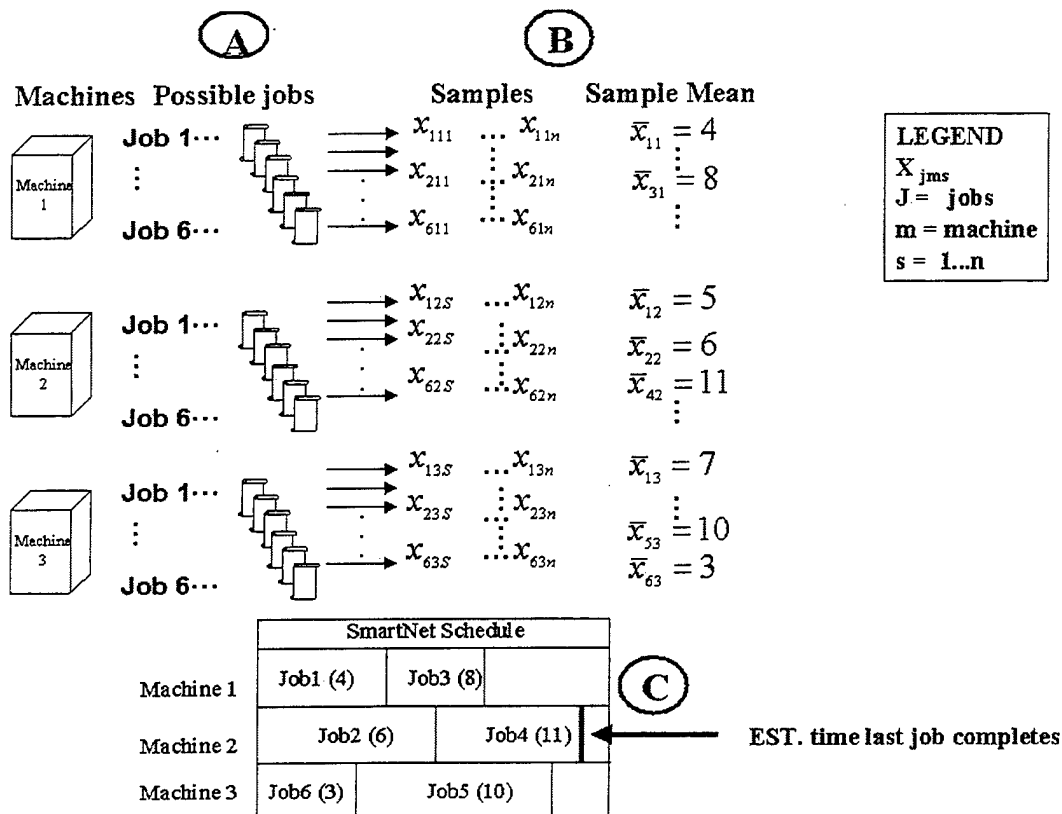


Figure 6: Scheduling Algorithms Using only the Estimated Mean

One such problem is shown below in Figure 7. In Figure 7, Part A, the (i^{th}, j^{th}) element of the matrix contains the average execution time of n runs of Job j on Machine m . In our example, the estimated runtimes for any of the jobs on any of the machines are exactly the same (six time units). Because of this, it does not matter where a job is scheduled. In Part B of Figure 6, Job #1 is scheduled for Machine #1, Job #2 is scheduled for Machine #2, and Job #3 is scheduled for Machine #3.

It is possible that all the scheduled jobs could run for less than or exactly equal to their expected runtimes. Since all the jobs finish either early or right on time, the expected time at which the last job completes is less than or equal to six and this schedule will perform well. Unfortunately, the job execution times can also finish after their expected execution time. In Part C of Figure 7, the actual job execution times are marked on their runtime distributions on the right side of the figure as tick marks. We can see that Job #1 finished at three time units, well before its estimated time to complete of six time units. This early finish does not have a negative effect on the system. Unfortunately, Job #2 finished at eight time units and Job #3 finished at seven time units after their expected times to complete (of six time units). This shows that the expected time at which the last job completes cannot possibly be six time units. In this situation, if the data provided by either Job #2 or Job #3 was time sensitive, say the end user needed the data at or before the expected completion time, he would not have that data. In addition, any subsequently scheduled jobs would be delayed. The second distribution curve, Part D of Figure 6, shows that even if the actual expected time for the last job to complete were known for 50% of the jobs run, the actual time at which the last job completes would be greater than the expected time. This delay would cause the schedule to be inaccurate resulting in potential problems for the system. The final distribution curve, Part E of Figure 7, shows that for good results, the expected time at which the last job completes would need to include roughly 95% of its associated distribution. This is a level of accuracy that MSHN needs.

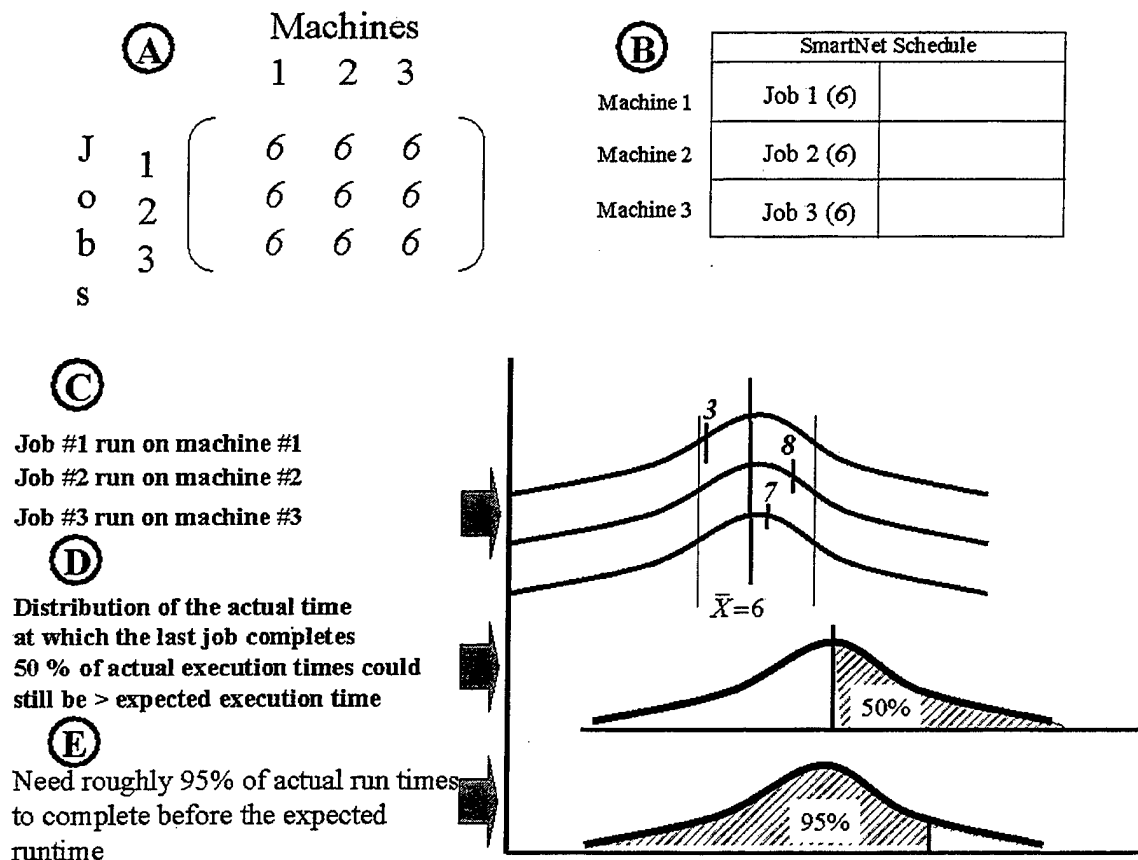


Figure 7: Job Runtime Distributions

The above example shows that current scheduling algorithms using only the mean in their calculations tend to underestimate the completion time of a schedule. These underestimates are likely to cause backlogs and inefficiencies in the system. For this reason, MSHN employs scheduling algorithms that use more information and produce better results. Stochastic scheduling algorithms have the potential to meet these requirements. They are algorithms that use higher moments and other distribution information to compute their schedules. Such algorithms will make MSHN more effective. In order to use such algorithms, MSHN must have some means of producing

higher order moments and distributions (i.e., of producing distribution statistics). This thesis is an initial attempt at producing such statistics.

THIS PAGE INTENTIONALLY LEFT BLANK

III. OVERVIEW OF PREVIOUS WORK

This chapter briefly describes how four different system tools conduct resource monitoring. Resource monitoring is an integral part of MSHN and most other Resource Management Systems (RMSs). RMSs, in general, need an agent to provide information on the availability of resources in the system so that applications can be scheduled. Tools such as Network Weather System also need an agent of sorts to monitor resources so that they can accurately predict resource performance. As discussed in earlier chapters, the agent used for resource monitoring in MSHN is the Client Library. The tools and RMSs described below are the Network Weather System (NWS), SmartNet, MSHN, DeSiDeRaTa, Jewel, CONDOR, and Odyssey.

A. RESOURCE MONITORING

Simply put, resource monitoring is a mechanism that permits you to be aware at all time of what resources you have and how much of each resource is available for use. In most instances this is easier said than done because of an ever rapidly changing environment. The following example helps illustrate the point. One of the many training exercises a Tank Company Commander (CO) is responsible for is that for an annual gunnery qualification. Generally speaking, the resources necessary to accomplish this mission are his fourteen tank crews, fourteen tanks, a tank gunnery range, ammunition, fuel, mechanics, and a medical squad.

If the company commander solely owned all of these resources, and had a birds eye view of them at all times, completing the gunnery qualification on time and with good results would be relatively easy. This, however, is not the case. Most, if not all, of these resources are shared in some way. For instance, there are several companies vying to use the same gunnery range, and higher level commands can take away personnel, equipment and supplies at any time. The CO, then, must employ his "agents" (First Sergeant, Training Non Commissioned Officer (TNCO), Platoon Leaders and Platoon Sergeants) to keep him current on resource availability so that he can make informed decisions. The agents record and report resource usage and availability to the CO. The CO can then use this information to change his plan accordingly and accomplish his mission to the best of

his ability using the reported available resources. In addition, and perhaps more importantly, all of the information collected by the CO's agents are used in an After Action Review (AAR) when the gunnery is complete. The AAR is conducted so that the CO and his staff can record and save lessons learned, good and bad, from the exercise. The lessons learned can then be applied to the planning and execution of the next annual gunnery in hopes of improving overall performance.

While this system of employing agents is good, it is not perfect. One of the biggest problems is getting accurate information to the CO in a timely manner. For example, the TNCO knows that the unit is running low on ammunition but waits until the last minute to tell the CO. In this situation, the unit will experience considerable downtime, because they must wait while more ammunition is brought out to the range. If, on the other hand, the TNCO notifies the CO of the ammunition problem in a timely manner, the CO can have the ammunition brought out sooner and have it waiting at the range. This same example can be applied to many of the above resources (fuel, supplies, etc.). Accurate reporting is also important. If the TNCO reports to the CO that the gunnery range is available for the two-week period that the CO wants to train, but in reality the range is only available for the first week, the gunnery will not be completed on time. Again, this situation also applies to more than just the range resource. For the most part, the CO's agents are both timely and accurate. The reason these agents are timely and accurate is because, in this case, the agents can physically be at the resource location and see and report exactly what the resource availability is without getting in the way (i.e., affecting the resource being measured). His agents also have several different lines of communication for reporting. One of the most important points in the example above is that the CO is getting near perfect information on his resources without suffering any mission performance degradation. This is a hard thing to accomplish in a distributed computing environment.

RMSs and tools that perform resource monitoring in a distributed computing environment, theoretically, work much in the same way as in the example above. These tools gather information about the resources in their environment and use the information for everything from scheduling to forecasting. One of the challenges that these tools must overcome is to monitor the information without adding overhead. For example, running the **ping** program is a way to measure network throughput between two machines. Upon

connected computers. When **ping** completes, it records the number of bytes sent and the round trip time, from which we can get an estimate of throughput. The problem with running **ping** is that it places an additional load on the resource being measured leading to inaccuracies. To get a better idea of how this challenge and others are overcome, the following sections describe four system application tools and how they measure resources.

1. Network Weather Service (NWS)

The NWS is an application designed to sense the performance of resources throughout its environment while, based on this information, providing forecasts of the future performance of those resources [WOLS97]. NWS collects the resource information via three different sensors, a CPU sensor, a network link sensor, and a memory sensor. These sensors send their information to a subsystem where the data is preprocessed and passed to a database for use as inputs by up to three possible forecasting methods. The rest of this section describes how the sensors in NWS gather information.

As with most data gathering agents, the NWS's CPU and Network sensors aim to limit their intrusiveness when taking measurements, thus lessening the opportunity for hindering the performance of the applications that are executing.

The NWS CPU sensor uses a combination of techniques to measure CPU availability. First, two Unix system utilities, **uptime** and **vmstat**, are used. The CPU Sensor calls **uptime** with the one-minute variable and calculates the fraction of the CPU a process would get if it were run at that particular time. When the CPU Sensor calls the Unix system utility **vmstat**, the fraction of the CPU a process would get is calculated using a combination of idle time, user time and system time measurements. The authors of [WOSH97] point out that, while these utilities do not add a significant amount of overhead, each can leave out significant information that could lead to inaccurate measurements of available CPU. The example that the [WASH97] article uses is that neither utility can provide information on the priority of any of the currently running processes. The CPU Sensor is designed to make up for

these shortcomings by running a type of daemon that mimics a compute intensive program. For example, the available CPU can be calculated as the ratio of the observed occupied CPU to the execution (wall-clock) time of the daemon. The Sensor then compares the results from this process to the measurements of **uptime** and **vmstat** and takes the more accurate information of the group. The obvious problem of running the daemon is that this procedure adds overhead. For this reason, the daemon is not run as often as the Unix utilities. The CPU Sensor, at times, runs all three processes simultaneously. This serves a couple of purposes. First, when all three processes are executed at the same time, the daemon process is taken as fact. Second, the daemon process results are used to bias the Unix utility results in order to keep them more accurate until the next daemon is executed.

To keep the number of daemon executions to a minimum, the Sensor applies special heuristics. In general, as long as the Unix utility results are relatively stable, the daemon does not need to run as often. As the Unix utility's results vary significantly, the daemon must run to get the most accurate estimate of CPU availability. For a more detailed explanation of how the NWS CPU Sensor works, see [WOSH97].

As mentioned earlier, NWS also deploys a set of Network Sensors. The Network sensors work much in the same way as the daemon process described in the CPU Sensor section above. Unlike the CPU Sensor, there is only one possibility for taking network measurements in NWS. The single method of network measurement stems from the lack of available and consistent performance data between arbitrary machines.

NWS Network Sensors record three measurements, latency, throughput and effective throughput across a network link. Every machine in the NWS environment executes a copy of the NWS server. This server maintains a copy of all machines being monitored and the TCP port that the server is connected to. Each server at some point receives a token. The token is basically a permission slip to allow that server to conduct a sampling of a network link. The server selects a host from its list and sends a round trip single word packet to that host. When the packet returns, an estimation of latency is calculated by dividing the round-trip time by two. Once the latency

estimation is complete, the server sends another packet with a specified amount of data to the same host and times the transfer. The throughput can then be calculated by dividing the data size by the transfer time. Finally, effective throughput can be calculated by dividing data size by the result of subtracting the transfer time from the latency. At this point, all three measurements can be stored for later use. This technique can be intrusive so the designers of NWS organized the network sensors in a specific way to limit the number of samples that are taken. For specifics on the Network Sensor Hierarchy, see [WOSH97,WOLS97].

2. SmartNet

SmartNet is covered extensively in Chapter I. Therefore, in this section we will simply present the SmartNet Architecture, Figure 8 below, and briefly describe how SmartNet computes and uses the average runtimes of applications for the scheduling of subsequent runs.

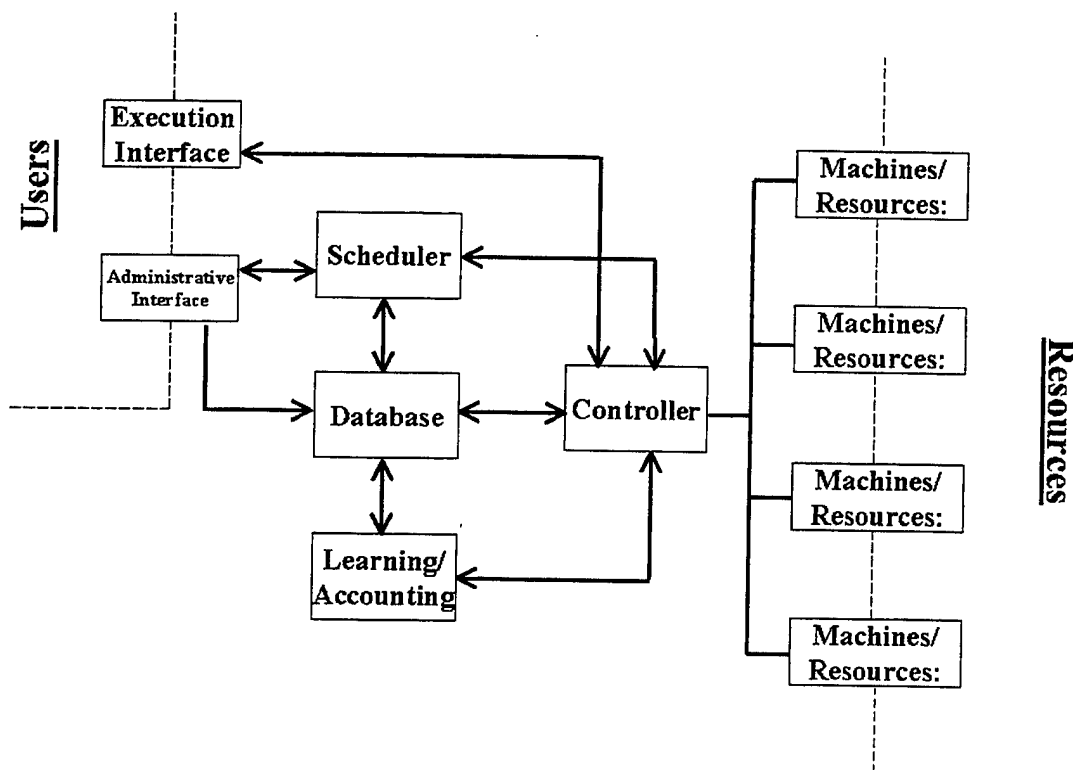


Figure 8: SmartNet Architecture From [KIDD96]

To understand how SmartNet computes the runtime estimates for subsequent runs of an application, consider the following example. A user submits an application to SmartNet, which includes as parameters an estimate of the expected runtime equal to two minutes and a weight equal to 80%. (The weight represents how sure the user is of the estimated runtime he is providing.) When the application starts, SmartNet starts a clock to measure wall clock time (WCT). When the application completes, the SmartNet timer stops and records the WCT. Let $WCT = 150$ sec for this example. SmartNet then computes a sample average running WCT and a sample standard deviation of the WCT. To keep the example simple, we ignore the sample standard deviation and let the sample average $WCT = 150$. The above information is stored in the SmartNet Database for later use by the SmartNet Scheduler. On subsequent runs of the above application, the SmartNet Scheduler would use the following equation to compute the input parameter.

Parameter = Weight x (Users Estimate Runtime) + (1 - Weight) x (Sample Average WCT)

Parameter = .80(120 sec) + (1-.80)(150 sec)

Parameter = 96 sec + 30 sec

Parameter = 126 sec

Concluding our example, the SmartNet Scheduler would use 126 seconds as its input parameter.

3. DeSiDeRaTa

DeSiDeRaTa is a Defense Advanced Research Projects Agency (DARPA) and Naval Surface Warfare Center (NSWC) sponsored software development project at the University of Texas at Arlington. It has the goals of providing resource and Quality of Service (QoS) management for Dynamic, Scalable, Dependable ReaT-Time Systems, hence the name DeSiDeRaTa. In short, DeSiDeRaTa aims to manage resources and real-time applications in a distributed shipboard computing systems domain. The difference between DeSiDeRaTa and previous works is that it accounts for complex

features such as variable periods, sporadic processes, priorities, fault management and scalability [DESI].

Of interest to us is how DeSiDeRaTa conducts resource monitoring. DeSiDeRaTa has several components to manage and monitor resources and applications in its environment. Figure 9 below is the author's interpretation of how the components work together to gather the needed resource information and make appropriate adjustments.

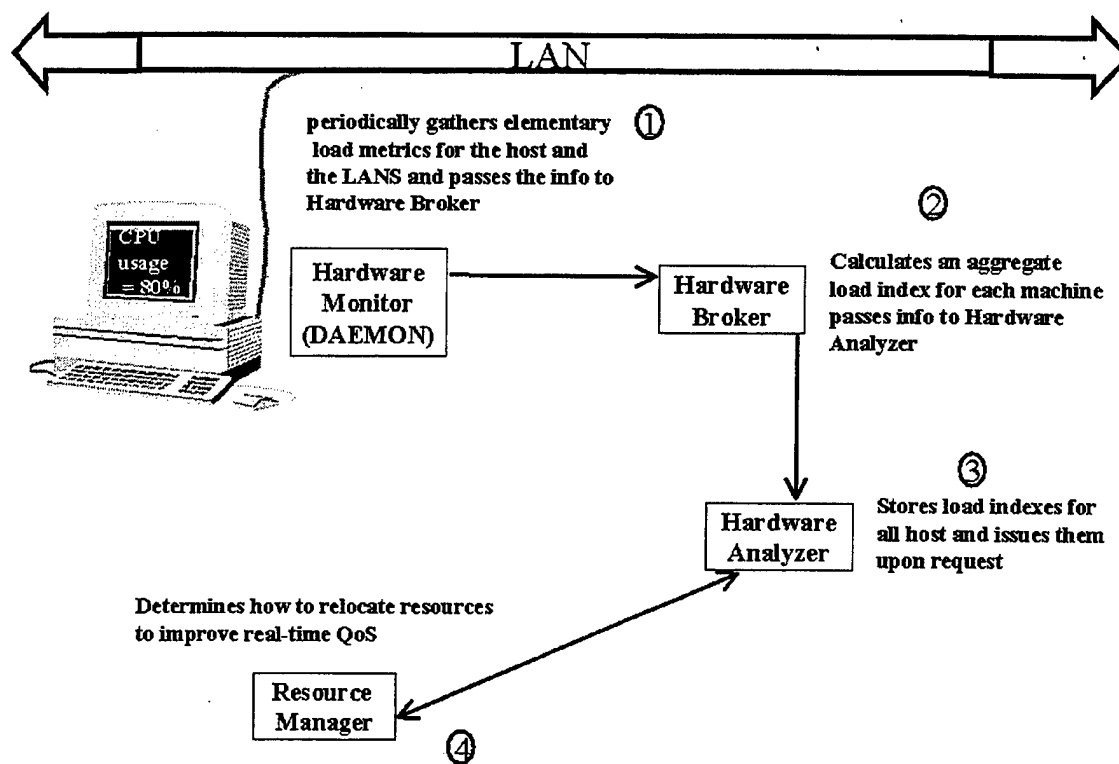


Figure 9: Resource Information Flow of DeSiDeRaTa

To monitor hardware in its domain, DeSiDeRaTa employs a daemon on each host. This daemon is a hardware monitor that periodically collects elementary load metrics for the host and the LAN. Examples of the above metrics are CPU queue length, free memory and context switches. The metrics collected by the Hardware

Monitor are passed to the Hardware Broker. The Hardware Broker uses the metrics to calculate an aggregate load index for each machine. The Hardware Broker then updates the Hardware Analyzer with the load index. The analyzer then serves as a database for this information. Finally, the Resource Manager can use this information to determine how best to re-allocate resources to improve real-time QoS. To get a complete description of the other DeSiDeRaTa components, see the DeSiDeRaTa manual [DESI].

4. Condor

Condor is a system developed at the University of Wisconsin-Madison that locates where there are machines with idle CPU cycles in its environment and schedules jobs to execute on them. Figure 10 shows how Condor is organized.

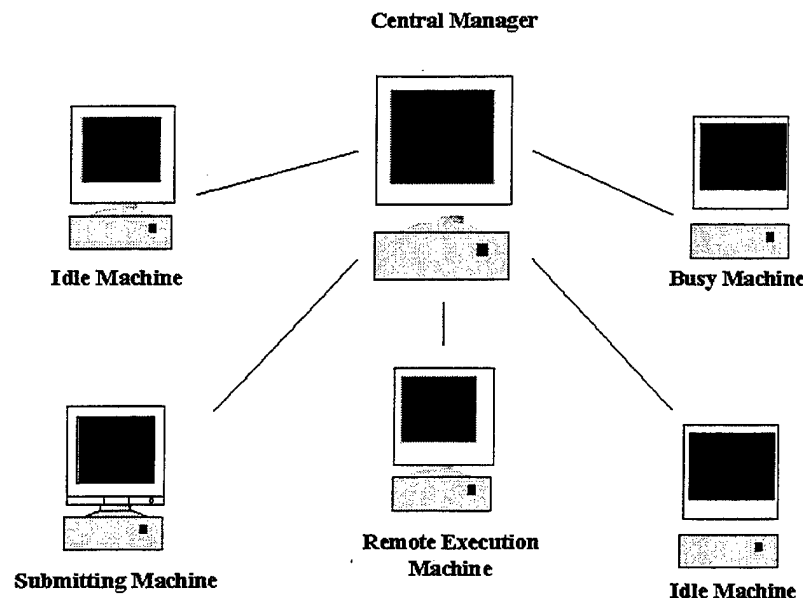


Figure 10: The Condor Pool From [SCHN98]

In the Condor system, jobs are submitted to the Central Manager where they are queued up until an idle machine is available. In Figure 10, the machines labeled “Idle Machine” are available for use by any job in the Condor system. The machine labeled “Remote Execution Machine” has a job assigned by the central manager executing on it. The machine labeled “Submitting Machine” is where the job that is executing on the Remote Execution Machine originated.

Resource monitoring in the Condor system is similar to resource monitoring in MSHN. In fact, some of the ideas for the MSHN Client Library (CL) came from Condor, in particular, the idea of wrapping system calls and statically linking jobs with libraries. Condor, like MSHN, operates by intercepting system calls made by user applications. The Condor Library, operating on the Remote Execution Machine, intercepts the application system call and sends it back to the Submitting Machine where the call can be processed. Once the system call has completed processing, the results are passed back to the Condor Library on the Remote Execution Machine, where the library returns the results to the user’s system call. In addition to tracking application resource requirements, Condor also employs daemons to constantly monitor system activity. These daemons send signals to the Condor Library, letting it know that the workstation is no longer available. When the library receives the signal, it writes information to a “checkpoint file” and terminates the process. The “checkpoint file” is a file that contains state information so that when the process starts on another machine, it can restart where it terminated. For further details on how the Condor system works, see [LITZ97]

5. Odyssey

A team from Carnegie Mellon University designed the software platform Odyssey to manage resources (i.e., the available network bandwidth and network quality) in a mobile computing environment. Odyssey is a system that interfaces with an application, determines the necessary amount of resources needed by an application, and if those resource requirements can not be met, allows the application to adapt and execute in a degraded mode. A similar situation occurs if more of a resource becomes available; the application can adapt and execute in an upgraded mode. The ability of an

application to adapt depends heavily on Odyssey's ability to monitor the resources in its environment. The following paragraph is dedicated to explaining how Odyssey monitors resources in its environment.

Odyssey can interface with its applications in two distinct ways. First, if the source code is readily available, then it (the source code) can be modified to interface directly with Odyssey components (see Figure 11). Second, if the source code is not available, then Odyssey employs a module called "cellophane" that seamlessly transforms application request into Odyssey objects. These techniques are key for resource monitoring in Odyssey's. Figure 11 is a graphic representation of the Odyssey client Architecture from [NOBL97] and is useful in explaining how Odyssey works.

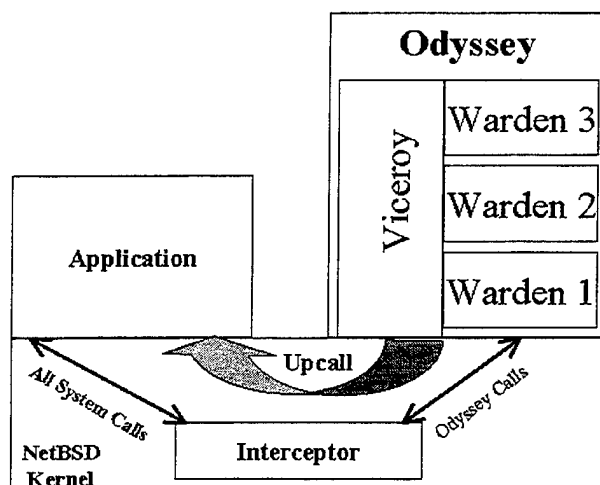


Figure 11: Odyssey Client Architecture After [NOBL97]

The Odyssey components consist of the Interceptor, the Viceroy and any number of Wardens (see Figure 11). The Interceptor intercepts all system calls made by the applications and passes them to the Viceroy. The Viceroy serves a couple of purposes. It is responsible for monitoring system performance and resource usage, and advises clients on adapting. The Viceroy gets an application's resource requirements when the application starts. The requirements are recorded and used for comparison against the reading of actual resource availability. The Wardens are responsible for communicating information between the Viceroy and clients, and they manage data types associated with their server. The server, in this case, is a remote program that provides data or processing for the mobile users [SCHN98]. For a full explanation and examples of how Odyssey works, see [NOBL97].

6. MSHN

MSHN's agent for monitoring resources is the Client Library (CL). The CL is a relatively unique way of monitoring resources and is explained in detail in Chapter V.

B. STOCHASTIC SCHEDULING

Stochastic scheduling is an extremely active research area and the work in this thesis closely meshes with it. Scheduling problems are inherently hard. They become even harder when uncertainty is introduced into the problem. For example, scheduling applications in a heterogeneous distributed network is hard even if the scheduler knows everything. Scheduling in a heterogeneous distributed network becomes even harder when there is some uncertainty in availability of resources and application or task times. There has been a significant amount of research in this area and this section was written simply to give reference to that work in case the reader is interested. The five, fairly recent, papers that we reference for this thesis are [ROSS91], [BECK00], [LIUR98], [BRES96] and [KAUF97].

C. SUMMARY

In this chapter, we discussed several existing tools that perform resource monitoring for one reason or another. We discussed how each of the tools conducts monitoring and some of the challenges that these tools had to overcome. For example, we saw that it was not entirely possible to eliminate the problem of intrusiveness in NWS, but the degree of intrusiveness could be limited. In Chapter V, we will see, in detail, how MSHN conducts resource monitoring and deals with the many challenges. In Section B, we discussed some of the current stochastic scheduling algorithms that exist and how they work.

IV. FORMAL DEFINITION OF THE PROBLEM

A. PROBLEM

The primary goal of this thesis is to study possible methods of aggregating data collected by the Client Library (CL) for later storage in the Resource Requirements Database (RRD) and the Resource Status Server (RSS). The use of statistics offers an excellent solution to our aggregation problem, and being able to accurately predict the distribution of the data allows MSHN to potentially make use of powerful distribution specific methods. There are several methods available to compute statistics and distributions. The problem is determining which method(s) provide the best (or most accurate) information. In addition, we want to obtain an initial estimate of how much, if any, of the raw data generated by the CL we need to maintain in the RRD and RSS.

B. APPROACHES TO SOLVING THE PROBLEM

For reasons discussed above and in Chapter IV, MSHN desires to be able to accurately predict the distribution of the data collected by the CL. Several “goodness of fit” tests exist that MSHN could use to predict these distributions. One of the most significant challenges in predicting a distribution is first determining which test to use. The tests chosen in this thesis are the Kolmogorov-Smirnov test, the Anderson-Darling test, and the Dirichlet Process. Each test is described in detail below. Included in the descriptions are the inherent advantages and disadvantages of the method and why the particular method was chosen for use in this thesis. The descriptions are followed by a brief summary of this chapter.

1. Kolmogorov-Smirnov Test (K-S test)

By definition, the K-S test examines whether a given sample of n observations is from a specified continuous distribution [JAIN91]. The K-S test measures the maximum deviation of the observed cdf from the theoretical cdf. This value is compared to a specific value found in tables giving the quantiles of the K-S distribution. If the maximum deviation of the observed cdf from the theoretical cdf, known as the K-S statistic, is less

than the table value, the sample is from the hypothesized distribution at the specified level of significance. Figure 12 shows the measured difference between a hypothesized distribution and an observed distribution. Note that the maximum deviation could be above or below the hypothesized distribution.

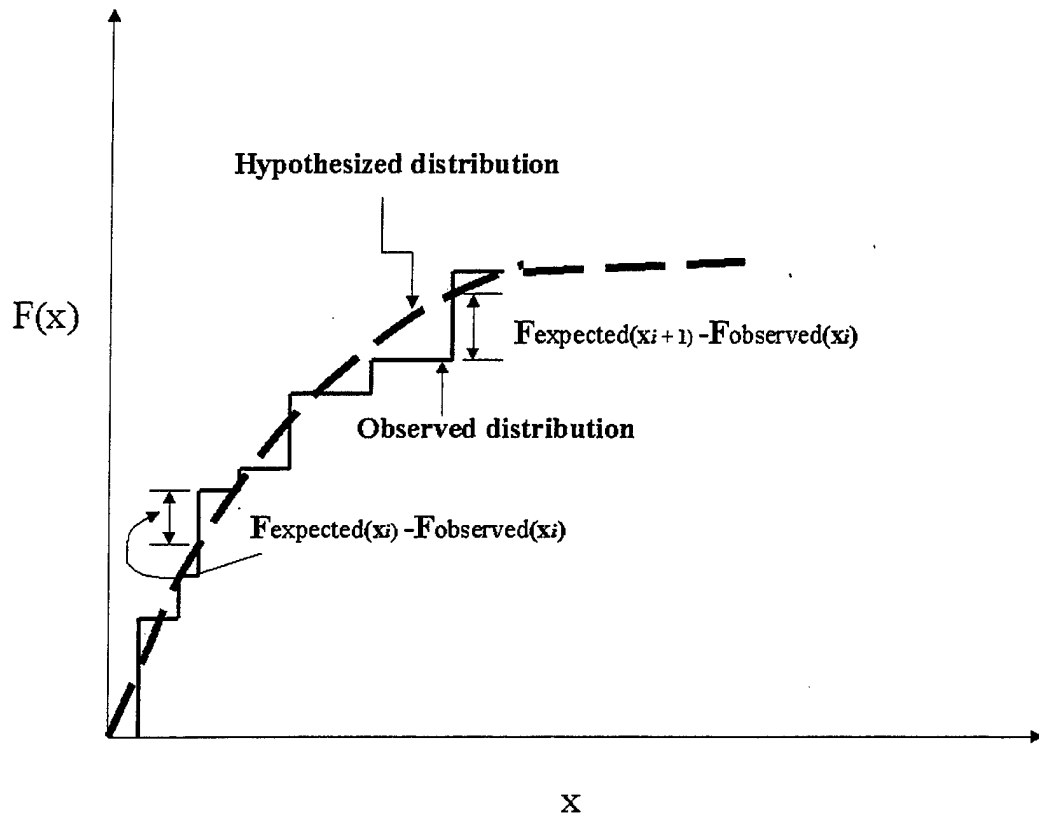


Figure 12: Hypothesized Distribution vs. Observed Distribution After [LAWK91]

The K-S goodness-of-fit test was chosen for use in this research for several reasons. Most significant are that the K-S test (1) does not require any grouping of the data and therefore no information is lost, (2) can be used for any sample size, and (3) tends to be more powerful than tests such as the chi-square test. Like all other “goodness of fit” tests researched, the K-S test has a possible drawback. According to [LAWK92], the original form of the K-S test is valid if and only if all of the parameters of the hypothesized distribution are known and the distribution is continuous. The data collected by the CL is continuous, but the population parameters are impossible to know. Therefore, the parameters must be estimated based on the sample data provided by the CL. Fortunately,

the K-S test was extended too particularly accommodate the estimating of parameters for the following four distributions: normal, log-normal, exponential and Weibull. The characteristics of previous data collected by predecessors to the CL leads us to believe that the distributions are likely exponential or Gaussian. Therefore, these are the distributions we will test our samples against. Testing the Gaussian and exponential distributions also eliminates the above drawback of the K-S test. To illustrate how the test works, a simple hypothetical example is provided below.

Suppose the CL collects the following wall-clock runtimes for ten executions of application Y. We want to test if the sample data is from a population with a Gaussian distribution where the parameters μ and σ^2 are unknown (i.e., the sample is from $N(\mu, \sigma^2)$). The data below was actually generated using Microsoft Excel's random number generator.

<u>Application Y Run #</u>	<u>Recorded Wall-Clock Time</u>
1	10.54
2	9.16
3	9.42
4	9.42
5	9.41
6	10.51
7	10.05
8	9.70
9	10.09
10	8.59

Table 2: Sample Data from a Normal Population

H_0 , our null hypothesis, is that our sample data is from a normal distribution.

$\left(\sqrt{n} - 0.01 + \frac{0.85}{\sqrt{n}}\right) D_n > C_{1-\alpha}$ is the formula used to test if our hypothesis should be accepted or rejected. If the above formula is true, we must reject H_0 in favor of H_a , the

alternative hypothesis (that the above data is not from a normal distribution). Our formula consists of three separate components, $\left(\sqrt{n} - 0.01 + \frac{0.85}{\sqrt{n}}\right)$, D_n , and $C_{1-\alpha}$.

$\left(\sqrt{n} - 0.01 + \frac{0.85}{\sqrt{n}}\right)$ is an adjustment to the test statistic to avoid having to use very large look up tables. D_n is the actual K-S statistic. The K-S statistic is the largest vertical distance between the hypothesized and empirical distribution. $D_n = \sup_x \left\{ \left| \hat{F}_n(x) - F(x) \right| \right\}$,

where $\hat{F}_n(x)$ is the empirical distribution and $F(x)$ is the hypothesized distribution. Because we are assuming that the above data comes from a population with a normal distribution, where the population mean (μ) and the population standard deviation (σ) are unknown (i.e., $F = N(\mu, \sigma)$), we must use a special case of the K-S test. We must first estimate both μ and σ by the sample mean ($\bar{X}(n)$) and the sample standard deviation ($S^2(n)$). The calculation to determine $F(x)$ is as follows; $F(x) = \Phi\left\{ \left[x - \bar{X}(n) \right] / \sqrt{S^2(n)} \right\}$, where Φ is the distribution function of the standard normal. Each data point, x , from our sample is input into the above formula, a decimal number is calculated, and that number is used to locate the value for $F(X_{(i)})$. From our example, for $F(X_{(1)})$, we start with $x = 8.59$, $\bar{X} = 9.689$, and $S^2 = 0.615$. $F(X_{(1)}) = \Phi\left\{ \left[x - \bar{X}(n) \right] / \sqrt{S^2(n)} \right\}$, substituting, we get $F(X_{(1)}) = \Phi\{-1.788\}$. Looking up -1.78 in the standard normal table, we get $F(X_{(1)}) = 0.0375$. The results for $F(X_{(i)})$ are listed below.

Application Y Run #	$\{[x - \bar{X}(n)]/\sqrt{S^2(n)}\}$	F(X _(i))
1	-1.788	0.0375
2	-0.861	0.1949
3	-0.454	0.0735
4	-0.438	0.3336
5	-0.438	0.3336
6	0.018	0.5714
7	0.587	0.7190
8	0.652	0.7422
9	1.336	0.9082
10	1.385	0.9162

Table 3: Results of F(X_(i)) i = 1 to 10

With the results from above, we can now finish our calculation of D_n . Because our data is in ascending order, we can compute $D_n^+ = \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - F(X_{(i)}) \right\}$ and $D_n^- = \max_{1 \leq i \leq n} \left\{ F(X_{(i)}) - \frac{i-1}{n} \right\}$. D_n is selected by taking the largest value of all of the D_n^+ and D_n^- computed. D_n^+ and D_n^- for $i = 1$ is calculated as an example.

$$D_n^+ = \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - F(X_{(i)}) \right\} = \left\{ \frac{1}{10} - 0.0375 \right\} = 0.0625 \text{ and } D_n^- = \max_{1 \leq i \leq n} \left\{ F(X_{(i)}) - \frac{i-1}{n} \right\} = \left\{ 0.0375 - \frac{1-1}{10} \right\} = 0.0375 \text{ at this point then, } D_n = 0.0625. \text{ Upon completion of the rest of}$$

the calculations for D_n , we find our result to be $D_n = 0.1664$. We now must apply the adjustment, $\left(\sqrt{n} - 0.01 + \frac{0.85}{n} \right)$, to our test statistic. Our formula is now

$$\left(\sqrt{10} - 0.01 + \frac{0.85}{10} \right) \times 0.1664 > C_{1-\alpha}, 3.2373 \times 0.1664 > C_{1-\alpha} \Rightarrow .5387 > C_{1-\alpha}. \text{ Looking up}$$

the modified critical value, $C_{1-\alpha}$, we see that 0.5387 is less than either 0.775, 0.819, 0.895, 0.955 or 1.035.

Case	Adjusted Statistic	$1-\alpha$			
		0.850	0.900	0.950	0.975
All parameters known	$\left(\sqrt{n}+0.12+\frac{0.11}{\sqrt{n}}\right)D_n$	1.138	1.224	1.358	1.480
$N(\bar{X}(n), S^2(n))$	$\left(\sqrt{n}-0.01+\frac{0.85}{\sqrt{n}}\right)D_n$	0.775	0.819	0.895	0.955
$\exp o(\bar{X}(n))$	$\left(D_n-\frac{0.2}{n}\right)\left(\sqrt{n}+0.26+\frac{0.5}{\sqrt{n}}\right)$	0.926	0.990	1.094	1.190

Table 4: Modified Critical Values for Adjusted K-S Test Statistics From [LAWK91]

This means that at significance levels of $\alpha = .15, .1, .05, .025$ and $.01$ (i.e., at 85%, 90%, 95%, 97.5% and 99% confidence levels), we accept that our data is from a normal distribution.

The K-S test can also be used to test that sample data came from a population with an exponential distribution. In this case, the hypothesized distribution is $\exp o(\beta)$, where β is unknown. We must estimate the parameter β by $\bar{X}(n)$. $F(x)$ is now defined to be the $\exp o(\bar{X}(n))$ distribution function and therefore, $F = 1 - e^{-x/\bar{X}(n)}$ for $x \geq 0$. Using the data from our example above, we can now test the H_0 (that the sample data is exponentially

distributed). The formula for D_n is equal to the larger value of

$$D_n^+ = \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - (1 - e^{-x/\bar{X}(n)}) \right\} \quad \text{and} \quad D_n^- = \max_{1 \leq i \leq n} \left\{ (1 - e^{-x/\bar{X}(n)}) - \frac{i-1}{n} \right\}$$

The adjustment for our statistic, D_n , is $(D_n - \frac{0.2}{n})(\sqrt{n} + 0.26 + \frac{0.5}{\sqrt{n}})$ from Table 2 above.

Based on our sample data, $D_n = 0.58793$, $n = 10$, and $(D_n - \frac{0.2}{n})(\sqrt{n} + 0.26 + \frac{0.5}{\sqrt{n}}) = 2.0334$.

From Table 4 above, we see that the null hypothesis must be rejected in favor of the alternative at the 85, 90, 95, 97.5 and 99% confidence levels, because the value for our adjusted statistic is larger than the critical values in Table 4.

2. Anderson-Darling Test (A-D test)

The Anderson-Darling test is similar to the K-S test in that it has a similar format and follows the same basic steps. In general, we must (1) determine the value of the A-D test statistic, which is denoted as A_n^2 , (2) make the appropriate adjustment according to the hypothesized distribution we are using, and (3) compare the computed value to the modified critical values for the adjusted A-D test statistic.

$$A_n^2 = \left(- \left\{ \sum_{i=1}^n (2i-1) [\ln Z_i + \ln(1 - Z_{n+1-i})] \right\} / n \right) - n$$

The A-D test has the ability to detect discrepancies in the tails of distributions, something the K-S test can not do. The A-D test is also a higher-powered test than the K-S test. For these reasons, the A-D test is included in this research. This particular form of the A-D test was selected because it allows the use of a much smaller critical value table (see [LAWK91]). The smaller critical value table made coding the test much easier. Examples of the A-D test, using the normal distribution and exponential distribution cases, are conducted below. In the first instance, we test the normal distribution. In this example, let $n = 10$, $A_{10}^2 = 0.02579$, and the adjustment $(1 + \frac{4}{10} - \frac{25}{100}) = 1.15$. The adjusted test statistic is then $(1.15)(0.02579) = 0.09266$. From Table 3 below, we can see that our hypothesis is accepted at all of the levels of significance.

Case	Adjusted Statistic	$1-\alpha$			
		0.900	0.950	0.975	0.990
All parameters known	A_n^2 for $n \geq 5$	1.933	2.492	3.070	3.857
$N(\bar{X}(n), S^2(n))$	$\left(1 + \frac{4}{n} - \frac{25}{n^2}\right) A_n^2$	0.632	0.751	0.870	1.029
$\exp o(\bar{X}(n))$	$\left(1 + \frac{0.6}{n}\right) A_n^2$	1.070	1.326	1.587	1.943

Table 5: Modified Critical Values for Adjusted A-D Test Statistics From [LAWK91]

Conducting the A-D test for the exponential case with the above sample data returns the following results: $n = 10$, $A_{10}^2 = 4.0704$, adjustment = $1 + \frac{0.6}{n}$. The adjusted test statistic is then $(4.0704)(1.06) = 4.3146$. From Table 5 above, we can see that our hypothesis is rejected at all of the levels of significance. In both the normal and exponential distribution cases, the A-D test and the K-S test properly accept and then reject the null hypothesis. Our next method takes a different approach to solving the problem.

3. Non-Parametric Bayesian Approach (Dirichlet process)

The Dirichlet process takes a slightly different approach than the above “goodness of fit” tests. Instead of testing if our data is from a particular distribution, we compute the actual distribution of the data. The formula for this process is

$$F_{new}(x) = \frac{\alpha}{\alpha+n} F_o(x) + \frac{n}{\alpha+n} \hat{F}(x)$$

The formula has three significant parts. $F_o(x)$ is a prior guess of the distribution (i.e., for $P(X \leq x)$) where, for example, $X \sim N(\mu=10, \sigma=2)$. $\hat{F}(x)$ is the empirical cumulative distribution function (cdf). This is an estimate of the true cdf, $F(x)$.

$$\hat{F}(x) = \frac{\{x_i \leq x\}}{n} \text{ where } n = \text{number of data elements.}$$

Finally, α is a weight put on $F_o(x)$, the prior guess. It is measured in number of observations worth. As an example, if we have 10 data elements and apply an α (weight) of 20 to our guess of the distribution, then what we are implying is that we are relatively confident that we have made an accurate guess as to what the real distribution is. In this example, our guess is worth 20 observations. If, on the other hand, we apply an α of 5 then we are implying that we are not very confident in our guess of what the underlying distribution is and, therefore, we apply a small weight to our guess. In this example, then, our weight is only worth 5 observations. In Figure 13 below, we show graphically how the Dirichlet process works. The guess for the distribution, $F_o(u)$, in this example is exponential. The alpha is very low because we are not certain that the guess made for the distribution is at all close to the actual distribution. Therefore, we are putting only a little emphasis on our guess. From Figure 13, we can see that the guess is indeed a bad one. The actual distribution, $F_\infty(u)$, is a normal distribution with a mean of 0 and a standard deviation of 1. As more data points are introduced to this process, the initial guess takes on less and less weight and the actual distribution is approached based on the empirical data. Making a good initial guess and heavily weighting it is beneficial in that the actual distribution can be reached more rapidly than if we make a bad initial guess. Figure 13 below shows a bad initial guess with small α . We can see from Figure 13 that our initial

guess for the distribution is a significant distance from the actual distribution. Even so, the estimated cdf will eventually converge to the actual cdf.

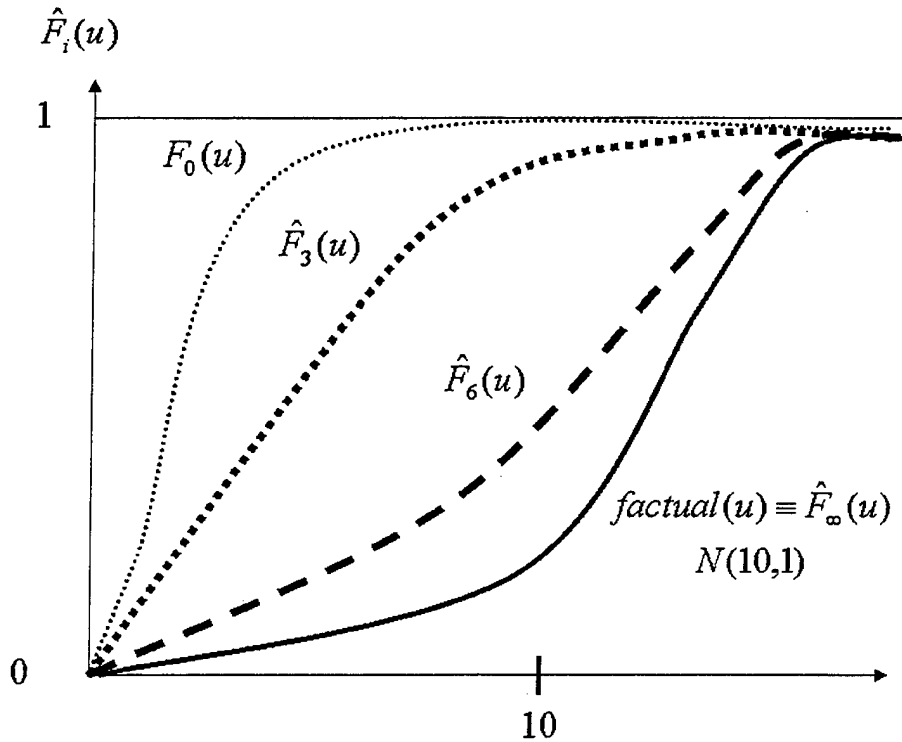


Figure 13: cdf Convergence Using the Dirichlet Process

C. SUMMARY

This chapter first discussed two common techniques that we use to determine if data from an application comes from a particular distribution. The third technique discussed, the Dirichlet process, actually converges to the real distribution of the data. It is important to note that there are many statistical approaches that could be implemented to solve our problem. These particular techniques were chosen for the reasons mentioned above.

V. MONITORING RESOURCES USING MSHN'SCLIENT LIBRARY

Monitoring resources in MSHN is done by using the MSHN Client Library (CL) with which each application is wrapped. This chapter describes how an application is wrapped, and how resources are monitored using the MSHN CL. The chapter also discusses some recent additions to the CL that provide finer grained data than did the previous CL version.

A. THE CLIENT LIBRARY

The MSHN CL was designed, implemented, and tested by Matthew Schnaidt, a 1998 graduate of the Naval Postgraduate School. Schnaidt's thesis entitled, "Design, Implementation, and Testing of MSHN's Resource Monitoring Library" [SCHN98] is the primary reference used for this chapter. For a complete understanding of the MSHN CL and its operation, see Schnaidt's thesis [SCHN98]. Recently, Shirley Kidd has modified the CL to include finer grained output information. A more detailed discussion of these modifications can be found in Section D of this chapter.

At a high level, the operation of the CL can be described as follows. An application to be run in the MSHN environment is first "wrapped" (as explained in the next section) by the CL. Once wrapped, the application is scheduled and eventually run on some machine in the MSHN environment. As the application runs, the CL records the application's resource usage (e.g., the amount of disk space, CPU time and network time used) and forwards the information to the appropriate MSHN component (e.g., the Resource Requirements Database). In the next section, we take a closer look at what it means to wrap an application, as wrapping is the key to the mechanism used to transparently monitor both the usage and status of resources in MSHN.

B. A WRAPPED APPLICATION

In this section, we contrast the normal process of compiling and linking Bison with one where Bison is wrapped with the MSHN CL.

Figure 14 below compares how an application, in this example Bison, is normally compiled (left) with how the same application is compiled and linked with the MSHN wrapper (right).

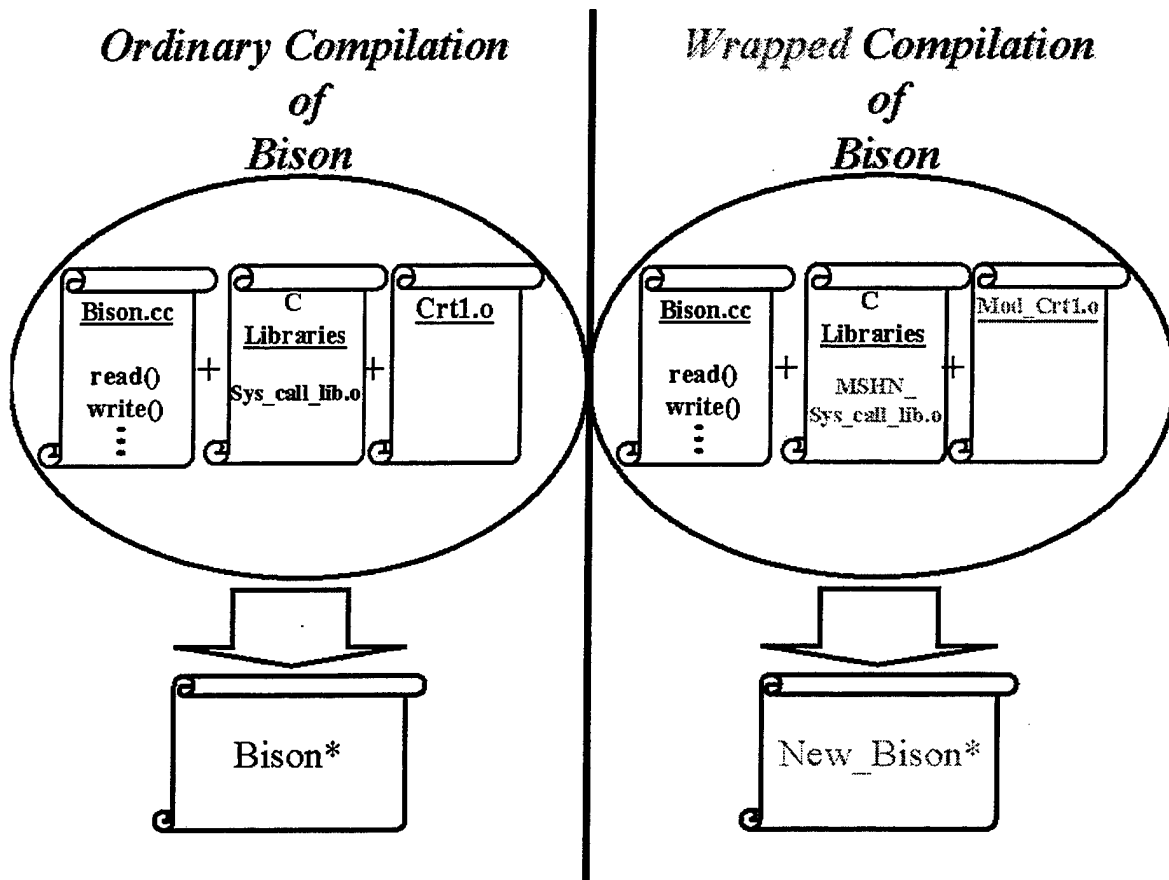


Figure 14: Ordinary Linking of Bison vs. Wrapped Linking of Bison

On the left side of Figure 14, the object file for Bison is linked with the appropriate libraries needed to define the many system calls made during Bison's execution. In Figure 14, we show `bison.cc` linking with the standard C libraries and the `crt1.o` file. When linking is complete, the executable file, **bison***, is created and the program can be used. Looking at the right side of Figure 14, we see that the process is the same, but the files that the object code is linked with are different. For the Client Library to produce the results we are looking for, e.g., the number of reads, the number of bytes read, the bytes written,

and the number of writes for both local and network disks, the application must link with modified C libraries and a modified crt1.o file. These modifications allow the MSHN wrapper to intercept the process' system calls, record the appropriate resource usage and apparent resource status, and report this information to MSHN's Resource Requirements Database (RRD) and Resource Status Server (RSS). How the wrapper intercepts a system call is discussed in further detail in the following section.

C. MONITORING RESOURCES

As mentioned above, to monitor a resource in MSHN, the MSHN wrapper/CL must intercept certain system calls prior to the calls reaching the OS. How this interception is done is not important for this thesis, but if interested, the reader is referred to [SCHN98]. For this thesis, we need only understand the general concept of how a system call is intercepted and what happens subsequently. To aid in explaining the process, we refer to Figure 15, below. Figure 15 is an event flow diagram from Schnaidt's thesis [SCHN98] illustrating what happens when a wrapped application invokes a `read()` system call. This type of interception is invoked for those system calls that, once modified, can provide MSHN with resource usage or status measurements.

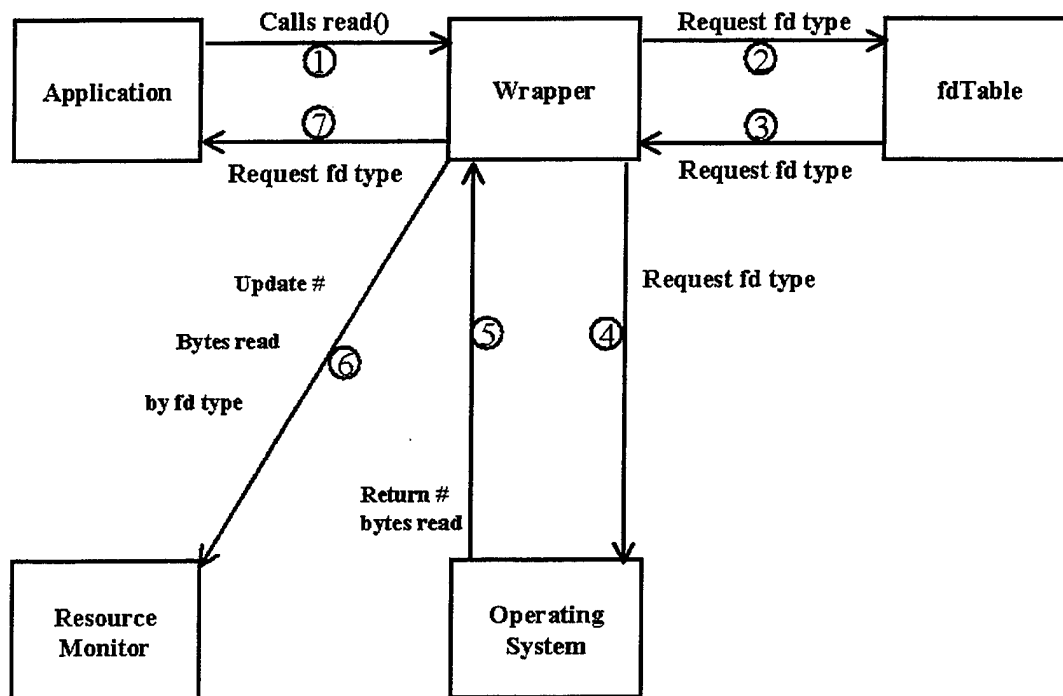


Figure 15: Event Flow Diagram for `read()` From [SCHN98]

The interception process is as follows. The application makes a call to `read()`. Before the call reaches the Operating System (OS), the MSHN wrapper intercepts it. The wrapper then looks in a file descriptor table (fdTable) maintained to determine the type of read being called. The wrapper then passes the `read()` to the OS. The OS returns the number of bytes read to the wrapper, the wrapper then updates the resource monitor and returns the size of the `read()`, along with the data read, back to the application. This is the general process that MSHN uses to monitor resource usage and status. The code inserted and executed between the application and the `read()` system call is part of the MSHN Client Library (CL). In the next section, we will look at the output of the prototype MSHN CL, and discuss some enhancements that have recently been made to the CL in order to provide to the other components of MSHN more fine-grained information.

D. CLIENT LIBRARY OUTPUT

In Schnaidt's thesis [SCHN98], MSHN investigators identified the following resource usage information as being important (i.e., worth recording) in aiding the MSHN Scheduling Advisor in making scheduling decisions: Local File I/O, Network File I/O, Terminal I/O, Network I/O, and statistics concerning Local Inter Process Communication (IPC). Resource requirement information output by the prototype implementation of the MSHN CL is shown in Figure 16.¹

```
<PROGTERMINATIONDATA>  
<APPWALLCLOCKRUNSEC>75.0117  
<SYSCPUSECS>1.04  
<USRCPUSECS>0.26  
<MAXRESSETSZ>0  
<UNSHAREDMEM>0  
<PAGEFAULTS>0  
<PHYSPAGENUM>192  
<VIRTUALPAGENUM>768  
<RESIDENTPAGENUM>620
```

Figure 16: Some Resource Requirement Information Output by Initial Prototype of MSHN CL

The value associated with SYSCPUSECS in Figure 16 above shows that, over the entire 75.0117 seconds of the program's execution, the program executed in kernel mode for only 1.04 seconds of CPU time. Similarly, the value associated with USRCPUTIME

¹ The initial CL prototype also aggregated and output additional information concerning the end-to-end communication resource status, but that information is ignored for simplicity in this thesis.

indicates that the program executed in user mode for only 0.26 seconds. For wise scheduling decisions, this information may not be fine grained enough.

In order to get the fine grain information required for this thesis, Shirley Kidd, a member of the MSHN staff, modified the MSHN wrapper. The current version of the MSHN CL now has the ability to report information about individual system calls, as well as information concerning the program's behavior between system calls. The MSHN CL records the wall-clock time, as well as the user CPU and system CPU times, between calls. Additionally for each call, the CL records the type of call, the arguments passed to as well as returned from the call, and the wall-clock time, and system and user CPU time spent in that call. Example output from this new version is presented in Figure 17, and an illustration of the meaning of this data is shown in Figure 18.

System Call	Sys Call Interval	Between Call Interval	UsrCpu Interval	SysCpu Interval
open	0.00569105	0.0000000	0.0000000	0.0000000
open	0.000182033	0.00786495	0.0000000	0.0000000
Read_loc_file	6.01426	0.00152004	0.01	0.07
Read_loc_file	5.89944	0.000403047	0.0000000	0.02
Read_loc_file	5.98077	0.00158107	0.0000000	0.02
Read_loc_file	6.17567	0.000952005	0.0000000	0.05
Close_loc_file	.0000499487	0.000360012	0.0000000	0.0000000
open	0.000108957	0.00252807	0.0000000	0.0000000
Write_term_io	0.000455022	0.00671804	0.0000000	0.0000000
Write_term_io	0.000250936	0.000324965	0.0000000	0.0000000
Read_term_io	10.2382	0.00679016	0.0000000	0.07
Write_term_io	0.000294089	0.000579	0.0000000	0.0000000

Figure 17: Example Output From the MSHN Client Wrapper (in sec)

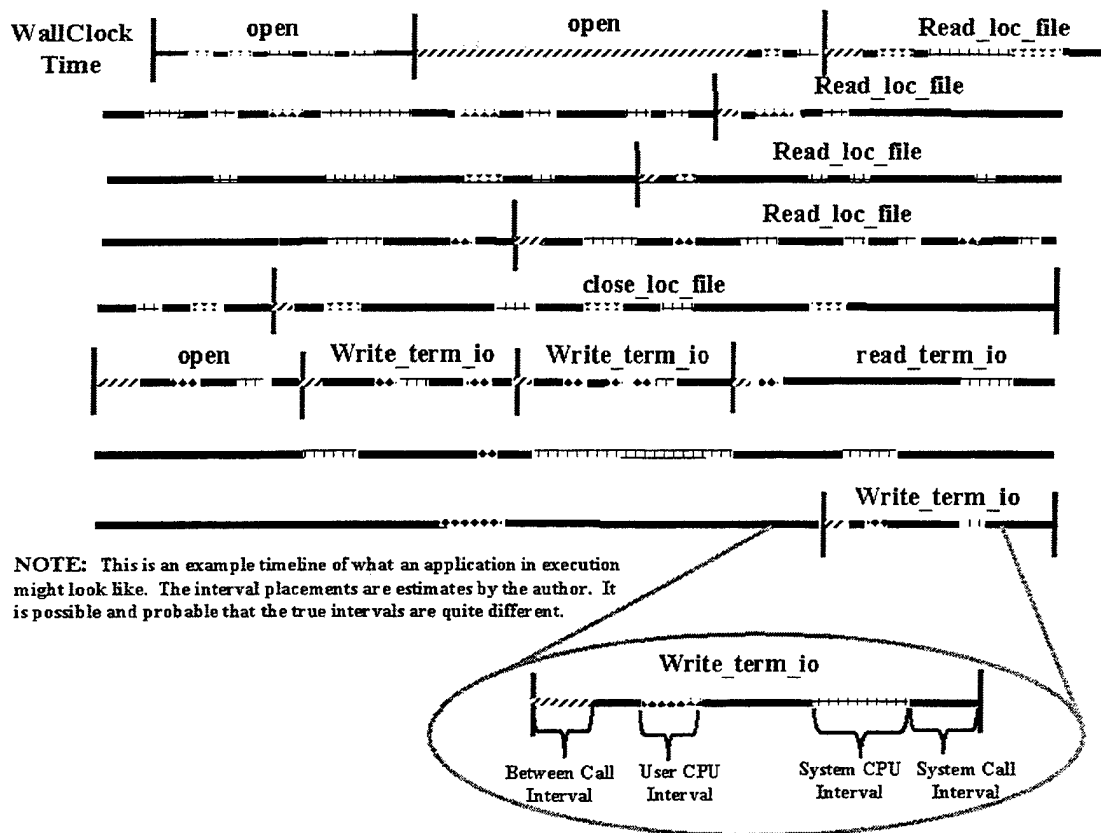


Figure 18: Example Execution Timeline of a Bison Application

Column 1 of Figure 17 shows every wrapped system call in the order it was made by the application, Bison. Reading across the rows of the table, one can see how time was consumed during that particular system call. Figure 18 is read from left to right, and top to bottom. It is a continuous timeline of the program Bison during execution. We took the times in Figure 17 and represented them in a timeline like fashion. The timeline, though accurately representing the data, is not precise in its fine detail. The author has estimated this fine detail, e.g., the exact location of the system CPU Interval in Write_term_io. The call-out in Figure 18, for example, depicts the time intervals of last wrapped system call made. Using both Figure 17 and Figure 18 we can see that, of the total 0.000294089 seconds spent in the Write_term_io system call, some very small portion was spent in

user and system CPU space. It is important to note that even though Figure 17 shows 0.0000 seconds for several of the user CPU and system CPU intervals, these measurements are not correct. These intervals were so small that the timing routines used did not have the ability to measure the interval. A full explanation of the importance of this data can be found in Chapter VI.

E. SUMMARY

In this chapter, we discussed how applications being scheduled to run in the MSHN environment are wrapped. We discussed how the CL monitors the resources an application is using, and finally, we discussed the contents of the CL's output and some of the new capabilities recently added to the CL.

VI. EXPERIMENT

This chapter describes the experiment designed to determine if the output of the statistical applications used in this thesis will prove useful for future stochastic scheduling algorithms. Section A describes (1) how the statistical applications were designed and implemented using Java, (2) the Bison input file used for this experiment, and (3) how we wrapped, ran and extracted the appropriate data from the Bison application. Section B describes the experiment's methodology, and how and why the experiment was run. Section C presents the results of the experiment. Section D describes our analysis of the results and our conclusions. Section E is a summary of this chapter.

A. DESIGN OF THE EXPERIMENT

The design of this experiment is best explained in two phases: Phase 1 describes the wrapping of Bison, and the pre-processing of MSHN Client Library (CL) output. Phase 2 details the design and implementation of the statistical package developed for this thesis.

1. Wrapping and running the application and pre-processing CL output

This section describes Bison, the application we chose to wrap, and the process used to wrap it. The section also describes the pre-processing done to the MSHN CL output file prior to the execution of the experiment.

Bison is a general-purpose parser generator that converts a formal description of an LALR(1) context-free grammar into a C program that parses that grammar. We selected Bison as the application to wrap because the Bison code was readily available.

Wrapping Bison was fairly simple. There are, however, a few comments worth noting. First, a full tutorial on wrapping system calls and wrapping applications is available in Schnaidt's thesis [SCHN98]. This tutorial should be completed in its entirety before attempting to wrap any executable. The tutorial first explains the process of wrapping a system call and then explains wrapping an application. Wrapping a system call and wrapping an application are very similar tasks; as such, it is easy to confuse the two. The second note worth mentioning is that the current version of the wrapper only

intercepts certain system calls (i.e., only the system calls that have been wrapped by the MSHN staff). If a user wants to get information about a certain system call, that call must first be wrapped. For example, if the user wants to obtain usage information for the system calls `fork()` and `exec()`, these calls have to first be wrapped. My final observation about wrapping an application is in reference to the many libraries that a typical application must link with. Often errors will appear after all of the steps for wrapping an application have been completed and the `make` command is invoked. In the author's experience, these errors occur because some of the necessary libraries needed for linking are missing. To fix the errors, simply determine which libraries are missing and then add them to the linking path in the application's make file.

To ensure the wrapper did not change the output of the Bison application, we compared the resulting program generated by an input grammar, written by Gary Stone (a Ph.D. student at the Naval Postgraduate School), prior to wrapping Bison with one generated by a wrapped version of Bison. The resulting output files were identical. The only difference between the two runs was that the execution times were significantly different. The wrapped version of Bison took approximately 24 times longer than the unwrapped version, 5 seconds for the unwrapped version and 120 seconds for the wrapped version. The added overhead comes from additional code added to the wrapper to output the collected fine-grained data to the terminal. The added overhead would not occur under normal use of the MSHN CL as the code would be optimized and the output data cached.

After determining that the wrapped Bison application was working properly we obtained a much larger grammar file to use in our test. The grammar file chosen was the `parse.y` file found in GCC. We chose this grammar file because it produces much more data than that used in the initial comparison test. This particular file ran for 635.58 seconds and produced the output found in Appendix A. While all of the data in Appendix A could be useful, we are only interested in the "Between System Call" and "System Call" intervals that produced over 50 data elements.

Rather than sifting through the MSHN CL and manually counting and extracting the intervals that had over 100 entries Ramesh Mantri, a programmer on the MSHN staff, wrote a Perl script that parsed out the numerical data needed. A sample of this data is shown below in Figure 19.

```
0.000476003
0.000561953
0.000468016
0.00049901
0.000468016
0.000550985
0.000764966
0.000491023
0.000452042
0.00051105
0.000433922
0.000495911
0.000429034
0.000586987
0.000467896
0.000584006
```

Figure 19: Example of Data Elements Extracted From MSHN CL Using Perlscript

The data elements in Figure 19 are now in a form useable by the applications designed for our tests in this thesis.

2. Designing and implementing the statistics application

This section describes (1) the statistical packages used in this thesis' experiments, (2) the implemented the packages, and (3) their validation.

As discussed in previous chapters, one of the goals of this thesis is to make the MSHN CL data more useful for other MSHN components. To accomplish this, we aggregate and describe this data using statistics.

Java was chosen as the language in which to write our applications, because it is an object-oriented language and has many built-in classes useful to us. The overall goal was to first build an object that contained an array of data elements as well as statistics describing that data. That object could then be passed to any or all of the three-distribution tests written for this thesis.

Our experiment needed to obtain the following descriptive statistics for our data:

- Sum
- Mean
- Median
- Variance
- Standard deviation
- Largest data element
- Smallest data element
- Range

The above statistics are used in at least one of our distribution tests. As stated above, an object is created containing an array of data elements along with all of the above statistics. The data from the file generated by the wrapped CL fills the array. In subsequent versions our goal is to read the data from a stream. Next, the Java application invokes a method to sort the data elements. Finally, the application calls the appropriate statistical method. The code for creating the data object is in Appendix A.

Next, we built two distributions tests, the Komogorov-Smirinov (K-S) and Anderson-Darling (A-D) “goodness of fit” tests and a Dirichlet process. Chapter IV explains these applications in detail. Each of the tests/process takes the above data object as input.

The “goodness of fit” tests are simple hypothesis tests. In both the K-S and the A-D tests for this thesis, the null hypothesis is either that the data comes from a population with an underlying normal distribution or that the data comes from a population with an underlying exponential distribution. In each case, a normal and exponential statistic is computed, a numerical adjustment is made to the statistic and the result is compared to a value in a look-up table. Depending on where the adjusted statistic falls in the table, the null hypothesis is either accepted or rejected at a given level of significance.

The K-S and A-D “goodness of fit” tests were built using the equations and tables in [LAWK91]. We chose those particular methods because they (Law & Kelton) reduced the size of the look-up tables with which the adjusted statistic is compared while saving a significant amount of time and cost nothing in accuracy. Both of the “goodness of fit” test applications were built in the same manner. Each test takes as input a data object. The data elements of the object are then used to compute the observed cumulative distribution function (cdf). We then look for the largest difference between the observed cdf and a hypothesized cdf. This difference then becomes our statistic. Before we can compare this statistic to the values in our table, we must make the appropriate mathematical adjustment to the statistic according to the formulas (see [LAWK91]). The adjustment is based on form of the hypothesized distribution. Comparing the computed result with the values in the table allow us to accept or reject the null hypothesis. The source code for these applications is in Appendix A.

In addition to the “goodness of fit” tests, we also built an application based upon a process that converges to the actual distribution of the data. The method, as discussed in Chapter IV, is called the Dirichlet Process. The source code for this method is also in Appendix A. Presently, the Dirichlet Process built for this thesis can only determine the distribution for one data element at a time. Future work is needed to add a loop to compute the distribution for every data element in a particular set of data.

B. EXPERIMENT METHODOLOGY

The experiment for this thesis was designed to show that the above statistical tests and processes can provide useful information to the MSHN components. By useful, we mean that we can either accept or reject that the underlying distribution is from a particular family of distributions. Whether the tests accepted or rejected the null hypothesis, i.e., that the data is from a particular distribution, that information can be provided to and potentially improve the performance of subsequent (stochastic) scheduling algorithms. In addition if the tests reject the null hypothesis, those distributions can be eliminate in the RSS as being associated with the resource/data.

For this experiment, Bison is wrapped using the MSHN CL and a parse.y file, extracted from GNU C Compiler (GCC) is run. When the Bison application finishes, the MSHN CL output is run through our Perl program to extract the needed data elements. Once the data elements are in a file, data objects are created and there elements tested to see if those elements come from a population with an underlying normal distribution, an underlying exponential distribution, or neither.

C. RESULTS

Upon completion of execution of our wrapped version of Bison and running our Perl script program the MSHN CL provided us with the data elements found in Appendix B. For each of the six data sets we created data objects and computed the descriptive statistics. The descriptive statistics for each data set are located in Appendix B immediately after its data set. Using the descriptive statistics and the data set as input we ran the two goodness-of-fit tests. The results of these tests are in Appendix C.

Tables 6, 7 and 8 below show the data, the descriptive statistics of the data (computed using our statistical methods), and the “goodness of fit” test results of (computed using the “goodness of fit” tests). Figure 20 shows the descriptive statistics and a histogram of the data computed using MINITAB².

² MINITAB® is a statistical software package that was developed over 25 years ago to make data analysis easier. The version used in this thesis is version 12, ©1998

0.0001320	0.0001349	0.0001349	0.0001330	0.0001360
0.0001321	0.0001309	0.0001370	0.0001351	0.0001320
0.0001320	0.0001320	0.0001340	0.0001310	0.0001321
0.0001329	0.0001340	0.0001321	0.0001320	0.0001301
0.0001330	0.0001320	0.0001329	0.0001400	0.0001340
0.0001321	0.0001320	0.0001310	0.0001370	0.0001330
0.0001310	0.0001340	0.0001321	0.0001321	0.0001340
0.0001330	0.0001340	0.0001330	0.0001961	
0.0001329	0.0001310	0.0001329	0.0001329	
0.0001409	0.0001330	0.0001330	0.0001340	

Table 6 : “Write Remote File Between Call Interval” Data Captured using the MSHN Client Library

The Data object has 47 elements in its array

DESCRIPTIVE STATISTICS

Minimum = 1.30057E-4

Maximum = 1.96099E-4

Range = 6.604200000000002E-5

Sum = 0.006326914999999999

Mean = 1.3461521276595742E-4

median = 1.32918E-4

Variance = 8.848993495374656E-11

Std Dev = 9.406908894729796E-6

Skewness = 6.334653681690928

Kurtosis = 42.011280023941055

95% CI = 2.7184671756673306E-6 conducted with sample StdDev NOT population StdDev

Table 7: Descriptive Statistics for WRFBCI Data

“Write Remote File Between Call Interval” Results

TEST: K-S Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted K-S test statistic 2.48335644275209026 is greater than the modified critical value 1.035. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: K-S test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted K-S test statistic 4.422383945811191 is greater than the modified critical value 1.308. Therefore, reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

TEST: A-D test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted A-D test statistic 11.454589644697467 is greater than the modified critical value 1.029. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: A-D test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted A-D test statistic 20.68485876216289 is greater than the modified critical value 1.943. Therefore reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

Table 8: " Goodness Of Fit " Test Results using Our “Goodness Of Fit” Application

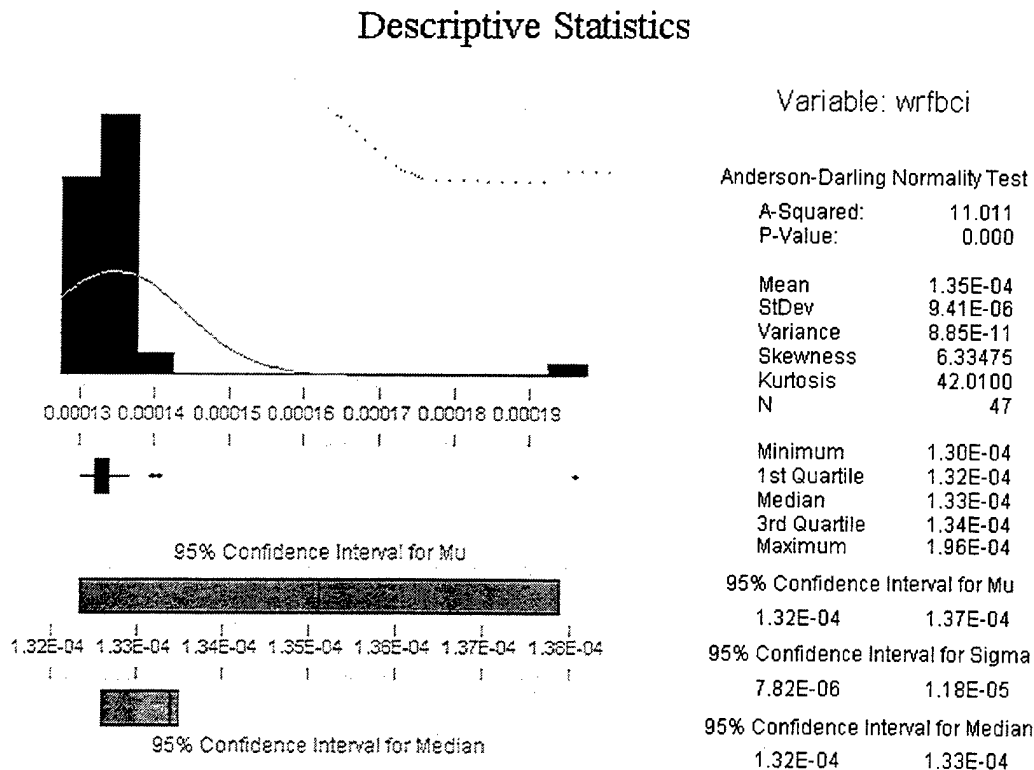


Figure 20: Descriptive Statistics and Histogram of WRFBCI Data Computed using Minitab

The data in Table 6 above is one of six different sets of data obtained from the MSHN Client Library (CL) after running a wrapped version of Bison. Each of the 47 data elements in Table 6 represent the time interval between each “write to a remote file” system call. The set of data in Table 6 is only a representative sample and is presented for discussion simply because the number of data elements are neither too large nor too small.

The descriptive statistics generated by this thesis’ statistical applications are presented in Table 7. In MSHN, these statistics will be stored in the Resource Requirements Database (RRD) for possible use by other MSHN components such as the MSHN Scheduling Advisor (SA). Future work must be done to update these statistics as soon as new data arrives.

With the information in Table 7, the “goodness of fit” tests can now be run to fit the data to a distribution. The results of our “goodness of fit” test are presented in Table 8.

As discussed in Chapter IV, we attempt to “fit” our data to two distributions, the normal distribution and the exponential distributions. In addition, we built two “goodness of fit” tests to check for each type of distribution.

In this experiment, both the Kolmogorov-Smirnov (K-S) and the Anderson-Darling (A-D) tests rejected the null hypothesis in favor of the alternative. The computed statistic for the K-S test is 2.4833 and the critical value, the value that we are comparing against, is 1.035. Clearly, 2.4833 is larger than 1.035 and, therefore, the null hypothesis is rejected. The computed Anderson-Darling (A-D) statistic is 11.4545 and the critical value is 1.029. Here too the statistic is larger than the critical value and the null hypothesis is rejected in favor of the alternative. Both tests rejected the null hypothesis at the largest possible Confidence Interval (CI), 99%.

The outcome is similar when testing the “fit” for the exponential distribution. The computed K-S statistic for the exponential distribution for this experiment is 4.4223. The critical value using a 99% CI is 1.308. Again, the computed statistic, 4.4223, is larger than the critical value, 1.308, and the null hypothesis is rejected in favor of the alternative. In the case of the A-D test, the computed A-D statistic for the data, 20.68, is significantly larger than the critical value, 1.943, and the null hypothesis is rejected in favor of the alternative. Like the tests for normality, these tests failed at the 99% CI.

Figure 20 shows Minitab’s descriptive statistics and histogram of our “write remote file between call interval” data. For all intent and purposes, the descriptive statistics computed by Minitab and computed using our application are identical, and thus, validate our descriptive statistics application. The histogram is presented to let the reader see that our “goodness of fit” tests are accurate in rejecting the null hypothesis. In Figure 20, Minitab also reports an A-D statistic for a normal distribution. Therefore, we can compare our adjusted A-D statistic for a normal distribution, 11.4545, with Minitab’s actual A-D statistic distribution, 11.011. The two statistics are relatively close. The reason for the difference is that the method we used actually computes an adjusted A-D statistic. The adjustment to the A-D statistic allows us to use a modified set of critical values ([LAWK91]. Had we not made any adjustments to the A-D statistics in our method the two statistics would have been exact, but we would have then had to use a complete A-D critical values table.

Appendices A-D show the results of the experiments performed on each of the six sets of data. In every instance, the null hypothesis was rejected in favor of the alternative. Therefore, our data does not come from either a population with an underlying normal distribution or a population with an exponential distribution. In fact, a couple of our data sets when plotted appear to have more than one mode. Upon further thought, the nature of the intervals measured, i.e., "Read Local File System Call Interval", "Write Remote File Between Call Interval", ... make them more likely to be multi-modal.

For example, the "Write Remote File Between Call Interval" does not pay any attention to what code is executing between the calls. A typical program will likely perform "Write Remote File Calls" for several different purposes and at several different locations in the code. The most frequent calls might be for writing data records. Less frequent calls might be for the attachment of header and trailer information to the data packet. The length of the interval between calls would differ depending upon the purpose of the code within which the calls are made, leading to multi-modal distribution for the "Between Call Interval". Additionally, multi-modal distributions could also result from relatively infrequent interrupts from Operating System (OS) maintenance calls causing the program to be swapped out (e.g., from file system maintenance routines).

D. CONCLUSIONS

Our experiments have shown that we can compute descriptive statistics of the data we receive from the MSHN CL and we can fit that data to any number of distributions. Other distributions can be "fit" by simply building the test as we did for the normal and exponential distributions. For the data sets studied in our experiments, our algorithms conclude that our data elements do not come from a population with either an underlying normal or an underlying exponential distribution.

We have discussed and partially built a Dirichlet process. The Dirichlet process computes an approximate distribution that converges to the actual distribution of a set of data. With more work, the Dirichlet process could be implemented into MSHN.

We now have the ability to store all of the descriptive statistics, the K-S and Anderson-Darling statistics in the Resource Requirements Database. The next step is to update these statistics as new data becomes available.

E. SUMMARY

In this chapter, we described the design of our experiment. We discussed how the application for calculating the descriptive statistics and the distribution tests work and were constructed. We presented and analyzed the results of the experiment and finally presented our conclusions.

VII. SUMMARY AND FUTURE WORK

A. SUMMARY

This thesis is broken down in to seven chapters. In the introduction we discussed Smart Net, the first scheduling framework for heterogeneous computing that considered, when scheduling jobs in its environment, both a jobs affinity for running on particular machines and the availability of machines. We introduced the Management System for Heterogeneous Network (MSHN), an improved management system designed to do research into how to provide all users with optimal Quality of Service. We discussed the scope of this thesis, its major contributions, and its organization.

In Chapter II, we discussed MSHN in great detail. We covered its evolution, goals, components, functionality and why we believe distribution statistics can be useful to MSHN. We focused primarily on the MSHN Client Library (CL) because it generated the data we needed for our experiments. We discussed why using only the mean in scheduling decisions is inadequate. And we posed our hypothesis: that we can provide more useful information than just the mean to future stochastic scheduling algorithms.

Chapter III covered previous works performed in the areas of resource monitoring and Stochastic Scheduling Algorithms. We looked at six different Resource Management Systems (RMSs) and tools, and discussed how they conduct resource monitoring. We discussed some of the pros and cons of the methods those RMSs and tools choose to use. We then discussed briefly Stochastic Scheduling Algorithms. These types of algorithms, or future algorithms like them, will make use of the information that we provide in this thesis.

Chapter IV described the formal definition of our problem and the approaches that we have chosen to solve this problem. We discussed the need for more useful information about the data that we get from the MSHN CL, and how that more useful information can be provided using statistics. We described why a set of data's underlying distribution can lead to more accurate scheduling when used in combination with Stochastic Scheduling Algorithms currently being developed. We then described two "goodness of fit tests," the Kolmogorov-Smirnov and the Anderson-Darling tests, chosen to see if a particular type of

data obtained from the MSHN CL is either normal or exponential. We point out that due to time constraints, these distributions are the only two we tested. With more time and research, several other tests could be implemented, making the MSHN scheduler even more effective. While these tests only confirm or deny that a particular distribution is associated with a set of data, our third approach eventually converges to the actual distribution. This approach is called the Dirichlet Process. Each of these three approaches is explained in great detail along with short examples for clarity.

Chapter V is a complete discussion on how the MSHN CL monitors resources in the MSHN environment. The first part of this chapter described how to wrap an application and gives an example of how we wrapped BISON. The second part of the chapter covered how MSHN intercepts and uses system calls to monitor resources. Finally, we provide output from the MSHN CL for both a wrapped and unwrapped application for comparison.

Chapter VI is our thesis experiment. This chapter described how we built the methods used to compute a data set's descriptive statistics. We discussed the design of the experiment and our experiment methodology. And finally, we presented our results and stated our conclusions.

Our final chapter, Chapter VII, summarizes this thesis and discusses possible future work.

B. FUTURE WORK

There are many possibilities for future work related to this thesis. Listed below are five areas that would be beneficial to both MSHN and other resource management systems in distributed environments.

1. Stochastic Scheduling Algorithms – While we know these algorithms exist and are a very active research area, we do not know the important details of how they work. Research needs to be conducted on what type of input such algorithms take. Once we know the input for such algorithms, the work done in this thesis can be modified and improved.

2. Implementing the code in this thesis in MSHN – The code used in this thesis was designed specifically to conduct our experiments. This code must be extended and made to interface with the MSHN CL, the MSHN RRD and the MSHN RSS.
3. Windowing mechanism for data – The code must also be made to read streams of data as opposed to reading data from a file. In addition to reading streams of data, a windowing mechanism must be built. A windowing mechanism prevents early data from anchoring more current data. Anchoring data is an important concept because it could possibly lead to wrong estimates of the underlying distributions.
4. Implementing tests for other distributions – This thesis tested for only two types of distributions, the normal and the exponential. Many other distributions exist and should be tested for. For example, the log-normal and Weibull distributions could be tested.
5. Multi-modal distributions – As it turned out, some of the distributions found in our experiments showed signs of being multi-modal. We must have some method of dealing with such distributions. Perhaps convolving or combining distributions would be possible.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: CODE FOR DATA OBJECT AND STATISTICAL TESTS

DataObject

```
/**
 * DataObject
 * @ Author MAJ Tom Cook
 * @ Version JDK 1.2
 * This program creates a Data Object. The Data Object consists of an array of
data and descriptive
 * statistics of that data. The descriptive statistics are: sum, min, max,
range, mean, variance,
 * standard deviation, skewness, kurtosis, 95% CI of mean, and the number of
data elements in the
 * array
 */

// _____

import java.io.*;
import java.util.*;

// _____

public class DataObject
{
// VARIABLE DECLARATIONS AND CONSTRUCTOR
// _____

    protected double data[];           // array to hold data
    protected double sum                = 0.0; // sum of data in array
    protected double variance          = 0.0; // variance of data in array
    protected double mean              = 0.0; // mean of the data in the array
    protected double median            = 0.0; // median of the data in the array
    //protected double mode             = 0.0; // mode of the data in the array
    protected double stdDev            = 0.0; // stdDev of the data in the array
    protected double min               = 0.0; // minimum of data in array
    protected double max               = 0.0; // minimum of data in array
    protected double range              = 0.0; // range of data in array
    protected double skewness          = 0.0; // skewness of data
    protected double kurtosis          = 0.0; // kurtosis of data POS = peak, NEG =
flat
    protected double confidence95      = 0.0; // 95% CI of the sample mean
    protected int    numDataElem       = 0;   // minimum of data in array
    protected long   timeObjBuilt;        // time the object was created
    protected Calendar rightNow;

    // constructor to create a dataObject object
    public DataObject()
    {
```

```

        String dataFromFile;                                // data from file read as a
string
        double dataElement;                                // convert data element to a
double
        boolean EOF = false;                                // End Of File

        data = new double [300];                            // new 300 element array
named data

        try
        {
            String dirName = "c:/data/Parse_yData";          // Directory path
            String fileName = "wrfbci.txt";                  // File Name
            File MyData = new File(dirName, fileName);

            BufferedReader DataIn = new BufferedReader(
                new FileReader(MyData));

            while (!EOF)
            {

                try
                {

                    dataFromFile = DataIn.readLine();
                    dataElement = Double.parseDouble(dataFromFile);

                    // if the array is full make another twice as big and copy the old into
the new
                    if (numDataElem == data.length)
                    {

                        double newData[] = new double [data.length * 2];
                        for (int idx = 0; idx < data.length; idx++)
                            newData [idx] = data[idx];
                        data = newData;
                    }

                    // fill the array
                    data [numDataElem] = dataElement;
                    numDataElem++;
                }

                // catch for an end of file exception
                catch (EOFException e)
                {
                    System.out.println (" In EOFException ");
                    EOF = true;
                }

                //catch for a null pointer exception
                catch (NullPointerException e)
                {
                    EOF = true;
                }
            }
        }

```

```

        // time object was built

        // sort the array in ascending order
        Arrays.sort(data,0,numDataElem);

        // Get the descriptive statistics of the data
        // System.out.println are debug statements

        min = min();
        //System.out.println("The minimum of the data is " + min);
        max = max();
        //System.out.println("The maximum of the data is " + max);
        range = range();
        //System.out.println("The range of the data is " + range);
        sum = sum();
        //System.out.println("The sum of the data is " + sum);
        mean = mean();
        //System.out.println("The mean of the data is " + mean);
        median = median();
        //System.out.println("The median of the data is " + median);
        //mode = mode();
        //System.out.println("The mode of the data is " + mode);
        variance = variance();
        //System.out.println("The variance of the data is " + variance);
        stdDev = stdDev();
        //System.out.println("The standard deviation of the data is " + stdDev);
        skewness = skewness();
        //System.out.println("The skewness of the data is " + skew);
        kurtosis = kurtosis();
        //System.out.println("The kurtosis of the data is " + kurtosis);
        confidence95 = confidence95();
        //System.out.println("The 95% CI of mean of the data is " + confidence95);
        // Method for printing the Data Object
        // PrintDataObj();

        // Close input stream
        DataIn.close();

    }

    // catch for stream creation exception
    catch (FileNotFoundException e)
    {
        System.err.println(e);
        return;
    }
    // catch for file read exception
    catch (IOException e)
    {
        System.err.println( "Error reading file " + e );
        return;
    }

return;

```

```

        } // end DataObject()

//
/**
 * get the minimum of the data in the Array
 * @ Author MAJ Tom Cook
 */
//

public double min()
{
    double minimum = 999999999.9;
    double tempMin = 0.0;

    for(int i = 0; i < numDataElem; ++i)
    {
        tempMin = data[i];
        if (tempMin <= minimum)
        {
            minimum = tempMin;
        }
    }
    return minimum;
} // end min

//
/**
 * get the maximum of the data in the Array
 * @ Author MAJ Tom Cook
 */
//

public double max()
{
    double maximum = -999999999.9;
    double tempMax= 0.0;

    for(int i = 0; i < numDataElem; ++i)
    {
        tempMax = data[i];
        if (tempMax >= maximum)
        {
            maximum = tempMax;
        }
    }
    return maximum;
} // end max

//
/**
 * get the range of the data in the Array
 * @ Author MAJ Tom Cook
 */
//

public double range()
{
    return Math.abs (max() - min());
} // end max

//
/**

```



```

* compute the sum of the data in the Array
* @ Author MAJ Tom Cook
*/
//
public double sum()
{
    double sumData = 0;

    for(int i = 0; i < numDataElem; ++i)
    {
        sumData += data[i];
    }

    return sumData;
} // end mean

//} // end Class DataObject
//
/**
* compute the mean of the data in the Array
* @ Author MAJ Tom Cook
*/
//
public double mean()
{
    double sumData = sum();
    return sumData/numDataElem;
} // end mean

//
/**
* compute the median of the data in the Array
* @ Author MAJ Tom Cook
*/
//
public double median()
{
    if (numDataElem <= 1)
    {
        return data[0];
    }

    if (numDataElem%2 == 0)
    {
        double answer = 0.0;

        answer = (data[(numDataElem/2) - 1] + data[numDataElem/2])/2;

        return answer;
    }
    else
    {
        return data[(numDataElem/2)-1];
    } // end median
}
//
/**

```

```

* compute the mode of the data in the Array
* @ Author MAJ Tom Cook
*/
//_____

// public double mode()
// {

// return count;
// } // end mode

//_____
/**
* compute the variance of the data in the Array
* @ Author MAJ Tom Cook
*/
//_____

public double variance()
{
    double sumXiMinusMeanSqr = 0.0;

    for(int i = 0; i < numDataElem; ++i)
    {
        sumXiMinusMeanSqr += Math.pow(data[i] - mean(),2);
    }

    return 1.0/(numDataElem-1) * sumXiMinusMeanSqr;
} // end variance.

//_____
/**
* compute the stdDev of the data in the Array
* @ Author MAJ Tom Cook
*/
//_____

public double stdDev()
{
    double variance = variance();
    return Math.sqrt(variance);
} // end stdDev

//_____
/**
* compute the skewness of the data in the Array
* @ Author MAJ Tom Cook
*/
//_____

public double skewness()
{
    double sumXiMinusMeanDivByStdDevCubed = 0.0;

    for(int i = 0; i < numDataElem; ++i)
    {
        sumXiMinusMeanDivByStdDevCubed += Math.pow(((data[i] - mean())/stdDev),3);
    }
}

```

```

    }

    return (numDataElem/((numDataElem-1.0) * (numDataElem-2.0))) *
sumXiMinusMeanDivByStdDevCubed;
    } // end skewness

// _____
/**
 * compute the kurtosis of the data in the Array
 * @ Author MAJ Tom Cook
 */
// _____

public double kurtosis()
{
    double sumXiMinusMeanDivByStdDev4th = 0.0;

    for(int i = 0; i < numDataElem; ++i)
    {
        sumXiMinusMeanDivByStdDev4th += Math.pow(((data[i] - mean)/stdDev),4);
    }

    return ((numDataElem*(numDataElem+1.0)/((numDataElem-1.0) * (numDataElem-2.0)
* (numDataElem-3.0))
        * sumXiMinusMeanDivByStdDev4th) - ((3.0 * (Math.pow(numDataElem-
1.0,2)))/
        ((numDataElem-2.0)*(numDataElem-3.0))));
    } // end kurtosis

// _____
/**
 * compute the confidence95 of the data in the Array
 * @ Author MAJ Tom Cook
 */
// _____

public double confidence95()
{
    double tableValue = 1.96;

    return tableValue * (stdDev/Math.sqrt(numDataElem-1.0));
} // end confidence95

// _____
/**
 * Print the data object
 * @ Author MAJ Tom Cook
 */
// _____

public void PrintDataObj()
{
    System.out.println();

    for(int i = 0; i < numDataElem; ++i)

```

```

        System.out.println(data[i]+ ", ");

        System.out.println ("\n The Data object has "+numDataElem+" elements in
it's array");
        // System.out.println ("\n and was created at "+timeObjBuilt+".");
        //System.out.println ("\n and was created at "+rightNow+".");
        System.out.println (" _____ ");
        System.out.println ("\n DESCRIPTIVE STATISTICS ");
        System.out.println (" _____ ");
        System.out.println (" Minimum   = "+min+" ");
        System.out.println (" Maximum   = "+max+" ");
        System.out.println (" Range     = "+range+" ");
        System.out.println (" Sum       = "+sum+" ");
        System.out.println (" Mean      = "+mean+" ");
        System.out.println (" median    = "+median+" ");
//      System.out.println (" mode       = "+mode+" ");
        System.out.println (" Variance  = "+variance+" ");
        System.out.println (" Std Dev   = "+stdDev+" ");
        System.out.println (" Skewness  = "+skewness+" ");
        System.out.println (" Kurtosis  = "+kurtosis+" ");
        System.out.println (" 95% CI    = "+confidence95+" NOTE: conducted with
sample StdDev NOT population StdDev ");
    } // end PrintDataObj

} // end Class DataObject

```

MyFileIOClass

```

/**
 * myFileIoClass
 * @ Author MAJ Tom Cook
 * @ Version JDK 1.2
 * This class is for all file IO done in this program except for the
initialization of
 * the data vector.
 */

// _____

import java.io.*;
import java.util.*;

// _____

/** Class to open and retrieve data from a file */

public class myFileIoClass
{

```

```
//


---


/** Method: readWeight
 * Return Value: double weight
 * Parameter: na
 * Purpose: This method reads a weight from a file
 */

public static double readWeight() {

    Vector alpha = new Vector ();
    double weight      = 0.0;
    double vectorSize = 0.0;
    int counter        = 0;
    boolean EOF = false; // boolean to determine the end of a file

    try
    {
        // Create a file object and an input stream object for the file
        String directory = "c:/Data/dirichlet/"; // Directory path
        String fileName = "weight.txt"; // File name
        File myData = new File(directory, fileName);
        BufferedReader testIn = new BufferedReader(new FileReader(myData));

        for (String nextLine = testIn.readLine(); nextLine != null;
             nextLine = testIn.readLine())
        {
            Double nextValue = Double.valueOf(nextLine);
            alpha.addElement(nextValue);
        }

        testIn.close(); // Close the input stream
    }

    catch(FileNotFoundException e) // Stream creation exception
    {
        System.err.println(e);
        System.exit(1); // End the program
    }

    catch(IOException e) // File read exception
    {
        System.err.println(" Error reading input file" + e );
        System.exit(1); // End the program
    }

    vectorSize = alpha.size();

    for (counter = 0; counter < vectorSize; counter++)
    {
        weight = (((Double)alpha.get(counter)).doubleValue());
    }

    //System.out.println(weight);
}

```

```

return weight;
} // End for mean

```

```

/** Method: readCdfValue
 * Return Value: double cdfValue
 * Parameter: na
 * Purpose: This method reads a cdf Value from a file
 */

public static double readCdfValue() {

    Vector cdfValue = new Vector ();
    double cdfVal = 0.0;
    double vectorSize = 0.0;
    int counter = 0;
    boolean EOF = false; // boolean to determine the end of a file

    try
    {
        // Create a file object and an input stream object for the file
        String directory = "c:/Data/dirichlet/"; // Directory path
        String fileName = "cdfValue.txt"; // File name
        File myData = new File(directory, fileName);
        BufferedReader testIn = new BufferedReader(new FileReader(myData));

        for (String nextLine = testIn.readLine(); nextLine != null;
             nextLine = testIn.readLine())
        {
            Double nextValue = Double.valueOf(nextLine);
            cdfValue.addElement(nextValue);
        }

        testIn.close(); // Close the input stream
    }

    catch(FileNotFoundException e) // Stream creation exception
    {
        System.err.println(e);
        System.exit(1); // End the program
    }

    catch(IOException e) // File read exception
    {
        System.err.println(" Error reading input file" + e );
        System.exit(1); // End the program
    }

    vectorSize = cdfValue.size();

    for (counter = 0; counter < vectorSize; counter++)
    {
        cdfVal = (((Double)cdfValue.get(counter)).doubleValue());
    }
}

```

```

    }

    //System.out.println(cdfVal);

    return cdfVal;
} // End for cdfValue

//


---


/** Method: readStdDevGuess
 * Return Value: double stdDevVal
 * Parameter: na
 * Purpose: This method reads a Standard Deviation estimation from a file
 */

public static double readStdDevGuess() {

    Vector stdDev = new Vector ();
    double stdDevVal = 0.0;
    double vectorSize = 0.0;
    int counter = 0;
    boolean EOF = false; // boolean to determine the end of a file

    try
    {
        // Create a file object and an input stream object for the file
        String directory = "c:/Data/dirichlet/"; // Directory path
        String fileName = "stdDevGuess.txt"; // File name
        File myData = new File(directory, fileName);
        BufferedReader testIn = new BufferedReader(new FileReader(myData));

        for (String nextLine = testIn.readLine(); nextLine != null;
             nextLine = testIn.readLine())
        {
            Double nextValue = Double.valueOf(nextLine);
            stdDev.addElement(nextValue);
        }

        testIn.close(); // Close the input stream
    }

    catch(FileNotFoundException e) // Stream creation exception
    {
        System.err.println(e);
        System.exit(1); // End the program
    }

    catch(IOException e) // File read exception
    {
        System.err.println(" Error reading input file" + e );
        System.exit(1); // End the program
    }

    vectorSize = stdDev.size();

```

```

        for (counter = 0; counter < vectorSize; counter++)
        {
            stdDevVal = (((Double)stdDev.get(counter)).doubleValue());
        }

        //System.out.println(stdDevVal);

    return stdDevVal;
} // End for readStdDevGuess()
//
//
/** Method: readMeanGuess
 * Return Value: double meanGuess
 * Parameter: na
 * Purpose: This method reads a mean estimation from a file
 */

public static double readMeanGuess() {

    Vector mean = new Vector ();
    double meanVal = 0.0;
    double vectorSize = 0.0;
    int counter = 0;
    boolean EOF = false; // boolean to determine the end of a file

    try
    {
        // Create a file object and an input stream object for the file
        String directory = "c:/Data/dirichlet/"; // Directory path
        String fileName = "meanGuess.txt"; // File name
        File myData = new File(directory, fileName);
        BufferedReader testIn = new BufferedReader(new FileReader(myData));

        for (String nextLine = testIn.readLine(); nextLine != null;
             nextLine = testIn.readLine())
        {
            Double nextValue = Double.valueOf(nextLine);
            mean.addElement(nextValue);
        }

        testIn.close(); // Close the input stream
    }

    catch(FileNotFoundException e) // Stream creation exception
    {
        System.err.println(e);
        System.exit(1); // End the program
    }

    catch(IOException e) // File read exception
    {
        System.err.println(" Error reading input file" + e );
        System.exit(1); // End the program
    }
}

```



```

    }

    vectorSize = mean.size();

    for (counter = 0; counter < vectorSize; counter++)
    {
        meanVal = (((Double)mean.get(counter)).doubleValue());
    }

    //System.out.println(stdDevVal);

    return meanVal;
} // End for readStdDevGuess()

} // End of myFileIoClass

```

MyStatsProgram

```

/**
 * Main
 * @ Author MAJ Tom Cook
 * @ Version JDK 1.2
 * This program creates a StatTest object. The StatTest Object consists of three
 * Statistical tests,
 * the Kolmogorov-Smirnov Test, the Anderson-Darling Test, and the Dirichlet
 * process. Each test takes a
 * Data Object as input and returns the results of the individual test.
 */

// _____

import java.io.*;
import java.util.*;

// _____

public class MyStatsProgram
{

    public static stdNormalTable STDNORMALTABLE = new stdNormalTable();

    // MAIN PROGRAM
    // _____

    public static void main(String args[]) throws IOException
    {

        DataObject myDataObject = new DataObject();
    }
}

```

```

        //StatTest.ksTestNormal(myDataObject);

        //StatTest.ksTestExpo(myDataObject);

        //StatTest.adNormal(myDataObject);

        //StatTest.adExpo (myDataObject);

        StatTest.dirichletTest (myDataObject);

    }

} // end MyStatsProgram

```

StatTest

```

/**
 * StatTest
 * @ Author MAJ Tom Cook
 * @ Version JDK 1.2
 * This program creates a StatTest object. The StatTest Object consists of three
Statistical tests,
 * the Kolmogorov-Smirnov Test, the Anderson-Darling Test, and the Dirichlet
process. Each test takes a
 * Data Object as input and returns the results of the individual test.
 */

// _____

import java.io.*;
import java.util.*;

// _____

public class StatTest
{
    // _____
    // _____

    /** Method: ksTestNormal
     * Return Value: double normal
     * Parameter: DataObject data
     * Purpose: This test allows one to test if a given sample of n observations is
from a
     * Normal distribution. It is based on the observation that the difference
     * between the OBSERVED Cumulative Distribution Function (CDF) and the EXPECTED
CDF should
     * be small. The K-S statistic  $D_n$  is the largest vertical distance between
 $F_n(x)$  and  $F^{\wedge}(x)$ 
     * for all values of  $x$  and is defined as  $D_n = \sup\{|F_n(x) - F^{\wedge}(x)|\}$ .
     *  $D_n$  is calculated as follows  $D_{n+} = \max \text{ for } (1 \leq i \leq n) \{(i/n) - F^{\wedge}(X(i))\}$ ,
     *  $D_{n-} = \max \text{ for } (1 \leq i \leq n) \{F^{\wedge}(X(i)) - ((i-1)/n)\}$ 
     *  $D_n = \max \{D_{n+}, D_{n-}\}$ 

```

```

*/

public static boolean ksTestNormal(DataObject data)
{
    double x          = 0.0;
    double z          = 0.0;
    double xMinusMean = 0.0;
    double tableValue = 0.0;
    double value       = 0.0;
    double maxPos      = 0.0;
    double maxNeg      = 0.0;
    double DnPos       = -9999999.0;
    double DnNeg       = -9999999.0;
    double Dn          = -9999999.0;
    double tempDn      = 0.0;
    double const4      = 0.01;
    double const5      = 0.85;
    int zrow           = 0;
    int zcolumn        = 0;
    double normal      = 0.0;
    boolean result;

    /** For Hypothesized Distribution N(pop mean, pop Var) both parameters
    unknown so we estimate them using Xbar(n) and Ssqrd(n) H0 is rejected if the
    adjusted statistic (sqrt n - 0.01 +(0.85/sqrt n)) Dn is > c' 1-alpha. where C' 1
    - alpha is a table look up. */

    for (int counter = 0; counter < data.numDataElem; counter++)
    {
        // get data element from array, subtract the mean, and divide the
        result by the
        // standard deviation
        // System.out.println(data.data[counter]);

        z = ((data.data[counter] - data.mean)/data.stdDev);
        // System.out.println (" z " + z);
        zrow = (int)(z * 10);
        zcolumn = (int)((z * 100)% 10);

        if (z < -3.49)
        {
            tableValue = .0002;
        }
        else if (z > 3.49)
        {
            tableValue = .9998;
        }
        else if (z < 0 & z >= -3.49)
        {
            value =
stdNormalTable.lookupValue(Math.abs(zrow),Math.abs(zcolumn));
            tableValue = 1.0 - value;
        }
        else
        {
            tableValue =
stdNormalTable.lookupValue(Math.abs(zrow),Math.abs(zcolumn));
        }
    }
}

```

```

        // System.out.println (" tableValue " + tableValue);

        maxPos = ((double)(counter + 1)/data.numDataElem) - tableValue;
        // System.out.println (" maxPos " + maxPos);
        maxNeg = tableValue - (((counter + 1) - 1)/(double)data.numDataElem);
        // System.out.println (" maxNeg " + maxNeg);

        tempDn = Math.max(maxPos,maxNeg);

        Dn = Math.max(Dn, tempDn);

    }

    //System.out.println (" tempDn = " + tempDn + "\n");
    //System.out.println (" Dn = " + Dn + "\n");

    normal = (Math.sqrt(data.numDataElem) - const4 +
const5/Math.sqrt(data.numDataElem))* Dn;

    //System.out.println (" normal = " + normal + "\n");

    result = CriticalValuesKsNormal(normal);

    System.out.println(result);

    return result;
} // end ksTestNormal
//
//
/** Method: CriticalValuesKsNormal
 * Return Value: void
 * Parameter:
 * Purpose: This method compares the passed value against the modified critical
 *          values in the table. The alpha values are: .15, .10, .05, .025,
and .01
 *          1 - alpha then equals .85, .90, .95, .975,
and .99
 *
 */

    public static boolean CriticalValuesKsNormal(double adjTestStat)
    {
        double critValDotEightFiveZero = 0.775;
        double critValDotNineZeroZero = 0.819;
        double critValDotNineFiveZero = 0.895;
        double critValDotNineSevenFive = 0.955;
        double critValDotNineNineZero = 1.035;

        if ( adjTestStat >= critValDotNineNineZero )
        {
            System.out.println( "\n The adjusted K-S test statistic " + adjTestStat + "
is greater than the modified critical value " + critValDotNineNineZero + "\n");

```

```

        System.out.println( " Therefore reject the null hypothesis that the data is
from an normal distribution using a 99% CI \n");
        return false;
    }
    else if ( adjTestStat <= critValDotNineNineZero & adjTestStat >
critValDotNineSevenFive )
    {
        System.out.println( " \n The adjusted K-S test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineNineZero + " but
greater than the modified critical value " + critValDotNineSevenFive + "\n");
        System.out.println( " Therefore fail to reject the null hypothesis; that the
data is from an normal distribution at using a 99% CI \n");
        return true;
    }

    else if ( adjTestStat <= critValDotNineSevenFive & adjTestStat >
critValDotNineFiveZero)
    {
        System.out.println( " \n The adjusted K-S test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineSevenFive + " but
greater than the modified critical value " + critValDotNineFiveZero + "\n");

        System.out.println( " Therefore fail to reject the null hypothesis; that the
data is from an normal distribution using a 97.5% CI \n");
        return true;
    }

    else if ( adjTestStat <= critValDotNineFiveZero & adjTestStat >
critValDotNineZeroZero)
    {
        System.out.println( " \n The adjusted K-S test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineFiveZero + " but
greater than the modified critical value " + critValDotNineZeroZero + "\n");
        System.out.println( " Therefore fail to reject the null hypothesis; that the
data is from an normal distribution using a 95% CI \n");
        return true;
    }

    else if ( adjTestStat <= critValDotNineZeroZero & adjTestStat >
critValDotEightFiveZero)
    {
        System.out.println( " \n The adjusted K-S test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineNineZero + " but
greater than the modified critical value " + critValDotEightFiveZero + "\n");
        System.out.println( " Therefore fail to reject the null hypothesis; that the
data is from an normal distribution using a 90% CI \n");
        return true;
    }
    else if ( adjTestStat < critValDotEightFiveZero)

        System.out.println( " \n The adjusted K-S test statistic " + adjTestStat +
" is less than the modified critical value " + critValDotEightFiveZero + "\n");
        System.out.println( " Therefore fail to reject the null hypothesis; that the
data is from an normal distribution using a 85% CI \n");
        return true;
    } // End for CriticalValuesKsNormal

//
/** Method: ksTestExpo

```

```

* Return Value: double result
* Parameter:
* Purpose: This test allows one to test if a given sample of n observations is
from a
* Exponential distribution. It is based on the observation that the
difference
* between the OBSERVED Cumulative Distribution Function (CDF) and tyhe
EXPECTED CDF should
* be small. The K-S statistic Dn is the largest vertical distance between
Fn(x) and F^(x)
* for all values of x and is defined as Dn = supremum{|Fn(x) - F^(x)|}.
* Dn is calculated as follows Dn+ = max for (1<=i<=n){(i/n) - F^(X(i))},
* Dn- = max for (1<=i<=n){F^(X(i)) - ((i-1)/n)}
* Dn = max {Dn+, Dn-}
*/

```

```

public static boolean ksTestExpo(DataObject data) {

    double oneMinusEtoExp = 0.0;
    double tableValue      = 0.0;
    double value            = 0.0;
    double maxPos           = 0.0;
    double maxNeg           = 0.0;
    double DnPos            = -9999999.0;
    double DnNeg            = -9999999.0;
    double Dn               = -9999999.0;
    double tempDn           = 0.0;
    double DnExpo           = 0.0;
    double const1           = 0.2;
    double const2           = 0.26;
    double const3           = 0.5;
    double expo             = 0.0;
    boolean result;

    //System.out.println(" I'm in ksTest \n");

    /** For Hypothesized Distribution expo(Beta) with Beta unknown
    * Beta is estimated by its MLE Xbar(n), and F^ is defined to be
    expo(Xbar(n)) dist Func
    * H0 is rejected if ((Dn - (0.2/n))((sqrt n + 0.26 + (0.5/ sqrt n)) >
    C' 1-alpha
    * where C' 1 - alpha is a table look up.
    */

    // DnExpo gets the value of the KS statistic calculated from the data

    for (int counter = 0; counter < data.numDataElem; counter ++)
    {
        oneMinusEtoExp = (1.0 - Math.pow(Math.E, (-
(data.data[counter])/data.mean)));
        //System.out.println ("1 - e** -x/mean = " + oneMinusEtoExp);

        maxPos = ((double)(counter + 1)/data.numDataElem) - oneMinusEtoExp;
        //System.out.println ("maxPos = " + maxPos);

        maxNeg = (oneMinusEtoExp - (double)((counter + 1) -
1)/data.numDataElem);
    }
}

```

```

        //System.out.println ("maxNeg = " + maxNeg);

        tempDn = Math.max(maxPos,maxNeg);

        Dn = Math.max(Dn, tempDn);
        //System.out.println ("Dn = " + Dn);
    }

    expo = (Dn - (const1/data.numDataElem)) * (Math.sqrt(data.numDataElem) +
const2 + (const3/Math.sqrt(data.numDataElem)));
    //System.out.println(" the adjusted statistic = " + expo + " \n");

    // CriticalValuesExpo(expo) looks up the result of the K-S statistic to
determine
    // whether to reject or fail to reject the null hypothesis at
    // alphah = .15, .1, .05, .025, and .01
    result = CriticalValuesKsExpo(expo);

    return result;
} // end ksTestExpo

// _____
// _____

/** Method: CriticalValuesKsExpo
 * Return Value: void
 * Parameter:
 * Purpose: This method compares the passed value against the modified critical
 * values in the table. The alpha values are: .15, .10, .05, .025,
and .01
 * 1 - alpha then equals .85, .90, .95, .975,
and .99
 */

    public static boolean CriticalValuesKsExpo(double adjTestStat)
    {
        double critValDotEightFiveZero = 0.926;
        double critValDotNineZeroZero = 0.990;
        double critValDotNineFiveZero = 1.094;
        double critValDotNineSevenFive = 1.190;
        double critValDotNineNineZero = 1.308;

        if ( adjTestStat >= critValDotNineNineZero )
        {
            System.out.println( "\n The adjusted K-S test statistic " + adjTestStat + "
is greater than the modified critical value " + critValDotNineNineZero + "\n");
            System.out.println( " Therefore reject the null hypothesis that the data is
from an exponential distribution using a 99% CI \n");
            return false;
        }
        else if ( adjTestStat <= critValDotNineNineZero & adjTestStat >
critValDotNineSevenFive )
        {
            System.out.println( " \n The adjusted K-S test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineNineZero + " but
greater than the modified critical value " + critValDotNineSevenFive + "\n");

```

```

        System.out.println( " Therefore fail to reject the null hypothesis; that the
data is from an exponential distribution at using a 99% CI \n");
        return true;
    }

    else if ( adjTestStat <= critValDotNineSevenFive & adjTestStat >
critValDotNineFiveZero)
    {
        System.out.println( " \n The adjusted K-S test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineSevenFive + " but
greater than the modified critical value " + critValDotNineFiveZero + "\n");

        System.out.println( " Therefore fail to reject the null hypothesis; that the
data is from an exponential distribution using a 97.5% CI \n");
        return true;
    }

    else if ( adjTestStat <= critValDotNineFiveZero & adjTestStat >
critValDotNineZeroZero)
    {
        System.out.println( " \n The adjusted K-S test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineFiveZero + " but
greater than the modified critical value " + critValDotNineZeroZero + "\n");
        System.out.println( " Therefore fail to reject the null hypothesis; that the
data is from an exponential distribution using a 95% CI \n");
        return true;
    }

    else if ( adjTestStat <= critValDotNineZeroZero & adjTestStat >
critValDotEightFiveZero)
    {
        System.out.println( " \n The adjusted K-S test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineNineZero + " but
greater than the modified critical value " + critValDotEightFiveZero + "\n");
        System.out.println( " Therefore fail to reject the null hypothesis; that the
data is from an exponential distribution using a 90% CI \n");
        return true;
    }
    else if ( adjTestStat < critValDotEightFiveZero)

        System.out.println( " \n The adjusted K-S test statistic " + adjTestStat +
" is less than the modified critical value " + critValDotEightFiveZero + "\n");
        System.out.println( " Therefore fail to reject the null hypothesis; that the
data is from an exponential distribution using a 85% CI \n");
        return true;

    } // End for CriticalValuesKsExpo
//
//


---


/** Method: adNormalTest
 * Return Value:
 * Parameter:
 * Purpose:
 */

public static boolean adNormal(DataObject data)
{

```



```

double ADNormalResult      = 0.0;
double ADNStatistic        = 0.0;
double Adjustment          = 0.0;
double s                   = 0.0;
double d                   = 0.0;
double q                   = 0.0;
double xMinusExpectedMean2;
double Isum                = 0.0;
double sum                 = 0.0;
double ADStat              = 0.0;
double value               = 0.0;
double value2              = 0.0;
double tableValue          = 0.0;
double tableValue2         = 0.0;
double tableResult         = 0.0;
double tableResult2        = 0.0;
double z                   = 0.0;
double zero                = 0.0;
int ten                    = 10;
int hundred                = 100;
int f                      = 0;
int p                      = 0;
int zrow                   = 0;
int zcolumn                = 0;
int qrow                   = 0;
int qcolumn                = 0;
boolean result;

for (int counter = 1; counter <= data.numDataElem; counter++)
{
    f = (2*counter - 1);

    z = ((data.data[counter - 1] - data.mean)/data.stdDev);
    //System.out.println (" z " + z);
    zrow = (int)(z * 10);
    zcolumn = (int)((z * 100)% 100);

    if (z < -3.49)
    {
        tableValue = .0002;
    }
    else if (z > 3.49)
    {
        tableValue = .9998;
    }
    else if (z < zero & z >= -3.49)
    {
        value = stdNormalTable.lookupValue(Math.abs(zrow),
Math.abs(zcolumn));

        tableValue = 1 - value;
    }
    else
    {
        tableValue = stdNormalTable.lookupValue(Math.abs(zrow),
Math.abs(zcolumn));
    }
}

```

```

    }

    tableResult = tableValue;
    //System.out.println (" tableResult = " + tableResult + "\n");

    p = ((data.numDataElem+1) - counter - 1);
    q = (data.data[p] - data.mean)/data.stdDev;
    //System.out.println (" The q = " + q + "\n");
    grow = (int)(q * ten);
    //System.out.println (" grow = " + grow + "\n");
    qcolumn = (int)((q * hundred)% ten);
    //System.out.println (" qcolumn = " + qcolumn + "\n");

    if (q < -3.49)
    {
        tableValue2 = .0002;
    }
    else if (q > 3.49)
    {
        tableValue2 = .9998;
    }
    else if (q < zero & q >= -3.49)
    {
        value = stdNormalTable.lookUpValue(Math.abs(grow),
Math.abs(qcolumn));

        tableValue2 = 1 - value;
    }
    else
    {
        tableValue2 = stdNormalTable.lookUpValue(Math.abs(grow),
Math.abs(qcolumn));
    }

    tableResult2 = tableValue2;
    //System.out.println (" tableResult2 = " + tableResult2 + "\n");

    Isum = f * (Math.log(tableResult) + Math.log(1.0 - tableResult2));
    sum = sum + Isum;
    //System.out.println (" sum = " + sum + "\n");
    //System.out.println (" f = " + f + "\n");
}

// compute the final statistic  $-(\text{sum})/n$  -n
ADStat = ((- sum)/data.numDataElem) - data.numDataElem;
//System.out.println (" sum = " + sum + "\n");
//System.out.println (" ADStat = " + ADStat + "\n");

// Calculates the adjusted Anderson-Darling statistic for a normal
distribution and
// stores the value in ADNormalResult
Adjustment = 1.0 + (4.0/data.numDataElem) -
(25.0/(data.numDataElem*data.numDataElem));
//System.out.println (" Adjustment = " + Adjustment + "\n");

```

```

        ADNormalResult = ADStat * Adjustment;
        //System.out.println (" ADNormalResult = " + ADNormalResult + "\n");

        // CriticalValuesAndersonNormal(ADNormalResult) looks up and determines
        whether to
        // reject or fail to reject the null hypothesis at alphah = .1, .05, .025,
        and .01
        result = CriticalValuesAndersonNormal(ADNormalResult);

        return result;
    } // end ADNormalTest
//
//


---




---


/** Method: CriticalValuesAndersonNormal
 * Return Value: void
 * Parameter:
 * Purpose: This method compares the passed value against the modified critical
 * values in the table. The alpha values are: .632, .751, .870,
and 1.029
 * 1 - alpha then equals .90, .95, .975,
and .99
 *
 */

    public static boolean CriticalValuesAndersonNormal(double adjTestStat)
    {
        double critValDotNineZeroZero = .632;
        double critValDotNineFiveZero = .751;
        double critValDotNineSevenFive = .870;
        double critValDotNineNineZero = 1.029;

        if ( adjTestStat > critValDotNineNineZero )
        {
            System.out.println( " \n The adjusted A-D test statistic " + adjTestStat + "
is greater than the modified critical value " + critValDotNineNineZero + "\n");
            System.out.println( " Therefore reject the null hypothesis that the data is
from an Normal distribution using a 99% CI");
            return false;
        }

        else if ( adjTestStat <= critValDotNineNineZero & adjTestStat >
critValDotNineSevenFive )
        {
            System.out.println( " \n The adjusted A-D test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineNineZero + " but
greater than the modified critical value " + critValDotNineSevenFive + "\n");
            System.out.println( " Therefore fail to reject the null hypothesis, that the
data is from an Normal distribution using a 99% CI");
            return true;
        }

        else if ( adjTestStat <= critValDotNineSevenFive & adjTestStat >
critValDotNineFiveZero)
        {

```

```

        System.out.println( " \n The adjusted A-D test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineSevenFive + " but
greater than the modified critical value " + critValDotNineFiveZero + "\n");
        System.out.println( " Therefore fail to reject the null hypothesis, that the
data is from an Normal distribution using a 97.5% CI, \n");
        return true;
    }
    else if ( adjTestStat <= critValDotNineFiveZero & adjTestStat >
critValDotNineZeroZero)
    {
        System.out.println( " \n The adjusted A-D test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineFiveZero + " but
greater than the modified critical value " + critValDotNineZeroZero + "\n");
        System.out.println( " Therefore fail to reject the null hypothesis, that the
data is from an Normal distribution using a 95% CI \n");
        return true;
    }
    else if ( adjTestStat <= critValDotNineZeroZero)

        System.out.println( " \n The adjusted A-D test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineFiveZero + "\n");
        System.out.println( " Therefore fail to reject the null hypothesis, that the
data is from an Normal distribution using a 90% CI \n");
        return true;

    } // End for CriticalValuesAndersonNormal
//
//
/** Method: AndersonDarlingExpo
 * Return Value: double ADStat
 * Parameter: Vector vector
 * Purpose: Returns the Anderson-Darling statistic for an exponential
distribution
 * based on the following formula;
 * ADStat =  $(-\{\text{sum from } i = 1 \text{ to } n (2i-1) [\ln Z_i + \ln(1 - Z_{n+1-i})]\}/n) - n$ 
 */

    public static boolean adExpo(DataObject data)
    {
        double oneMinuseRaisedToexp = 0.0;
        double oneMinuseRaisedToexp1 = 0.0;
        double oneMinuseRaisedToexp2 = 0.0;
        double one = 1.0;
        double Isum = 0.0;
        double sum = 0.0;
        double ADStat = 0.0;
        double AdjustedADStat = 0.0;
        int z = 0;
        int f = 0;
        boolean result;

        // loop through array to compute {sum from i = 1 to n (2i-1) [ln Zi + ln(1 -
Zn+1-i)]}
        for (int counter = 1; counter <= data.numDataElem; counter++)
        {

```

```

        f = (2*counter - 1);

        if (data.data[counter - 1] <= 0)
        {
            oneMinuseRaisedToexp = 0;
        }
        else
        {
            oneMinuseRaisedToexp = one - ( Math.pow(Math.E, - (data.data[counter -
1])/data.mean));
            //System.out.println (oneMinuseRaisedToexp);
        }

        z = ((data.numDataElem+1) - counter -1);
        //System.out.println (z);
        if (data.data[z] <= 0)
        {
            oneMinuseRaisedToexp2 = 0;
        }
        else
        {
            oneMinuseRaisedToexp2 = one - (one - (Math.pow(Math.E, -
(data.data[z])/data.mean)));
            //System.out.println (oneMinuseRaisedToexp2);
        }

        if (oneMinuseRaisedToexp <= 0 & oneMinuseRaisedToexp2 <= 0)
        {
            Isum = 0.0;
        }
        else if (oneMinuseRaisedToexp <= 0)
        {
            Isum = f*(Math.log(oneMinuseRaisedToexp2));
        }
        else if (oneMinuseRaisedToexp2 <= 0)
        {
            Isum = f*(Math.log(oneMinuseRaisedToexp));
        }
        else
        {
            Isum = f * (Math.log(oneMinuseRaisedToexp) +
Math.log(oneMinuseRaisedToexp2));
        }

        //System.out.println (" Isum = " + Isum + "\n");
        sum = sum + Isum;

        //System.out.println (" sum = " + sum + "\n");
        //System.out.println (" f = " + f + "\n");
    }

    // compute the final statistic  $-(\text{sum})/n$  -n
    ADStat = ((- sum)/data.numDataElem) - data.numDataElem;
    //System.out.println (" ADStat = " + ADStat + "\n");
    AdjustedADStat = (1.0 +(0.6/data.numDataElem))*ADStat;
    //System.out.println (" AdjustedADStat = " + AdjustedADStat + "\n");

```

```

        result = CriticalValuesAndersonExpo (AdjustedADStat);

        return result;
    } // End for AndersonDarlingExpo

//


---


/** Method: CriticalValuesAndersonExpo
 * Return Value: boolean
 * Parameter:
 * Purpose: This method compares the passed value against the modified critical
 *          values in the table. The alpha values are: 1.070, 1.326, 1.587,
and 1.943
 *          1 - alpha then equals .90, .95, .975,
and .99
 *
 */

    public static boolean CriticalValuesAndersonExpo(double adjTestStat)
    {
        double critValDotNineZeroZero = 1.070;
        double critValDotNineFiveZero = 1.326;
        double critValDotNineSevenFive = 1.587;
        double critValDotNineNineZero = 1.943;

        if ( adjTestStat > critValDotNineNineZero )
        {
            System.out.println( " \n The adjusted A-D test statistic " + adjTestStat + "
is greater than the modified critical value " + critValDotNineNineZero + "\n");
            System.out.println( " Therefore reject the null hypothesis that the data is
from an exponential distribution using a 99% CI");
            return false;
        }

        else if ( adjTestStat <= critValDotNineNineZero & adjTestStat >
critValDotNineSevenFive)
        {
            System.out.println( " \n The adjusted A-D test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineNineZero + " but
greater than the modified critical value " + critValDotNineSevenFive + "\n");
            System.out.println( " Therefore fail to reject the null hypothesis; that the
data is from an exponential distribution using a 99% CI\n");
            return true;
        }

        else if ( adjTestStat <= critValDotNineSevenFive & adjTestStat >
critValDotNineFiveZero )
        {
            System.out.println( " \n The adjusted A-D test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineSevenFive + " but
greater than the modified critical value " + critValDotNineFiveZero + "\n");
            System.out.println( " Therefore fail to reject the null hypothesis, that the
data is from an exponential distribution a 97.5% CI \n");
            return true;
        }

        else if ( adjTestStat <= critValDotNineFiveZero & adjTestStat >
critValDotNineZeroZero)
        {

```

```

        System.out.println( " \n The adjusted A-D test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineFiveZero + " but
greater than the modified critical value " + critValDotNineZeroZero + "\n");

```

```

        System.out.println( " Therefore fail to reject the null hypothesis, that the
data is from an exponential distribution using a 95% CI \n");

```

```

        return true;
    }

```

```

    else if ( adjTestStat <= critValDotNineZeroZero)

```

```

        System.out.println( " \n The adjusted A-D test statistic " + adjTestStat + "
is less than the modified critical value " + critValDotNineZeroZero + "\n");

```

```

        System.out.println( " Therefore fail to reject the null hypothesis, that the
data is from an exponential distribution using a 90% CI \n");

```

```

        return true;
    }

```

```

    } // End for CriticalValuesAndersonExpo

```

```

//

```

```

//

```

```

/** Method: dirichletTest

```

```

 * Return Value: Double answer

```

```

 * Parameter: Vector vector

```

```

 * Purpose: This method

```

```

 */

```

```

public static double dirichletTest(DataObject data)

```

```

{

```

```

    double sum          = 0.0;

```

```

    double vectorSize    = 0.0;

```

```

    double mean          = 0.0;

```

```

    double stdDev        = 0.0;

```

```

    double cdfValue      = 0.0; // need to read this from a file

```

```

    double weight        = 0.0; // read from file

```

```

    double lookUpValue   = 0.0; // read from a table

```

```

    double empiricalCdf  = 0.0;

```

```

    double answer        = 0.0;

```

```

    double Fobserved    = 0.0;

```

```

    double stdDevGuess   = 0.0; // read from a file

```

```

    double meanGuess     = 0.0; // read from a file

```

```

    double tableValue    = 0.0;

```

```

    double zero          = 0.0;

```

```

    double value         = 0.0;

```

```

    int Fobservedrow     = 0;

```

```

    int Fobservedcolumn = 0;

```

```

    // System.out.println("I'm in dirichletTest \n");

```

```

    // read weight (alpha) from file

```

```

    weight = myFileIoClass.readWeight();

```

```

    System.out.println("weight = " + weight);

```

```

    // read cdf value from file

```

```

    cdfValue = myFileIoClass.readCdfValue();

```

```

    System.out.println("cdfValue = " + cdfValue);

```

```

    // read weight (stdDevGuess) from file

```

```

    stdDevGuess = myFileIoClass.readStdDevGuess();

```

```

    System.out.println("stdDevGuess = " + stdDevGuess);

```

```

// read weight (meanGuess) from file
meanGuess = myFileIoClass.readMeanGuess();
System.out.println("meanGuess = " + meanGuess);

// Calculate the empirical CDF from your data
empiricalCdf = empiricalCdf(data);
System.out.println(" The empirical CDF for " + cdfValue + " = " +
empiricalCdf + "\n");

// compute F observed (x) = P( z <= ( (x-mean)/stdDev) )
Fobserved = ((cdfValue - meanGuess)/stdDevGuess); // must look up value for
Fobserved from table
System.out.println(" F observed = " + Fobserved + "\n");

Fobservedrow = (int)(Fobserved * 10);
Fobservedcolumn = (int)((Fobserved * 100)% 10);

    if (Fobserved < -3.49)
    {
        tableValue = .0002;
    }
    else if (Fobserved > 3.49)
    {
        tableValue = .9998;
    }
    else if (Fobserved < zero & Fobserved >= -3.49)
    {
        value = stdNormalTable.lookUpValue(Math.abs(Fobservedrow),
Math.abs(Fobservedcolumn));

        tableValue = 1 - value;
    }
    else
    {
        tableValue = stdNormalTable.lookUpValue(Math.abs(Fobservedrow),
Math.abs(Fobservedcolumn));
    }

    lookUpValue = tableValue;
    System.out.println (" lookUpValue for F observed = " + lookUpValue +
"\n");

// This is only good for
// N(mean = x, stdDev = y) must also check for Exponential...

// compute result of test
answer = ((weight/(weight + data.numDataElem))* lookUpValue) +
((data.numDataElem/(weight + data.numDataElem))* empiricalCdf);

System.out.println(" Dirichlet Result = " + answer + "\n");

return answer;
} // End for dirichletTest

```



```

// _____
// _____
/** Method: empiricalCdf
 * Return Value: double result
 * Parameter: Vector
 * Purpose: This method
 */

public static double empiricalCdf(DataObject data) {

    int counter                = 0;
    double value                = 0.0;
    double NumElementsLessThanEqx = 0.0;
    double cdfValue             = 3.0; //get this from a file
    double empiricalCdf         = 0.0;

    // compute the empirical CDF i.e if data = 2,3,4,5,10,20,16 then  $F^{\wedge}(x)$  (the
    empirical CDF)
    // where  $x = 6$ ,  $F^{\wedge}(6) = 4/7$  search vector and count all values  $\leq$  your  $x$ .

    // loop through the vector and count all the data that are less than the
    cdfValue
    for (counter = 0; counter < data.numDataElem; counter++)
    {
        value = data.data[counter];
        if( value <= cdfValue )
        {
            NumElementsLessThanEqx++;
        }
    }
    // empiricalCdf is defined by the number of data less than or equal to
    your cdfValue
    // divided by the number of data in the vector.
    empiricalCdf = (NumElementsLessThanEqx / data.numDataElem);

    return empiricalCdf;
} // End for empiricalCdf

// _____
} // end StatTest

```

StdNormalTable

```

/**
 * stdNormalTable
 * @ Author MAJ Tom Cook
 * @ Version JDK 1.2

```

```

    * This class constructs a standard normal distribution table. The class
    provides methods
    * to display the table and to look up values in the table when given the row
    and column
    */

```

```

// _____

```

```

import java.io.*;
import java.util.*;

```

```

/**
 * stdNormalTable
 * @ Author MAJ Tom Cook
 * @ Version JDK 1.2
 * This class constructs a standard normal distribution table
 */

```

```

// _____

```

```

public class stdNormalTable
{
    private static double [] [] cdfOfz;
    public stdNormalTable()
    {
        int rows = 35;
        int columns = 10;
        double value = 0.0;
        boolean EOF = false;

        cdfOfz = new double [rows][columns];

        // populate the array with values from c: data\StandardNormal
        try
        {
            // Create a file object and an input stream object for the file
            String directory = "c:/Data/Tables/";          //Directory path
            String fileName = "StandardNormalL.txt";      //File name
            File myData = new File(directory, fileName);
            BufferedReader testIn = new BufferedReader(new FileReader(myData));

            String nextLine = testIn.readLine();

            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < columns; j++)
                {
                    Double nextValue = Double.valueOf(nextLine);
                    cdfOfz[i][j] = nextValue.doubleValue();
                    //value = ((Double) cdfOfz [i][j]).doubleValue();
                    nextLine = testIn.readLine();
                }
            }

            testIn.close();          // Close the input stream

```

```

    }

    catch(FileNotFoundException e) // Stream creation exception
    {
        System.err.println(e);
        System.exit(1);           // End the program
    }

    catch(IOException e)           // File read exception
    {
        System.err.println(" Error reading input file" + e );
        System.exit(1);           // End the program
    }

    // displayTable(cdfOfz);

} // end table

```

// _____

```

/** Method: displayTable
 * Return Value:
 * Parameter:
 * Purpose: This method displays the standard Normal Table
 */

```

```

public static void displayTable (double [] [] array)
{
    int rows = 35;
    int columns = 10;

    System.out.println();

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            System.out.print(array[i][j] + " ");
        }

        System.out.println();
    }

    System.out.println();
} // End for displayTable

```

// _____

```

/** Method: lookUpValue
 * Return Value: double result
 * Parameter: Vector
 * Purpose: This method
 */

```

```

public static double lookUpValue(int row, int column)
{
    return cdfOfz[row][column];
}

```

```
    } // End for lookUpValue  
  
} // end StdNormalTable
```

APPENDIX B: MSHN WRAPPER OUTPUT: BISON APPLICATION

Read Local File Between Call Interval Data

0.0002379420	0.00657201	0.010213	0.00282001	0.00282204
0.0008879900	0.00290203	0.00279593	0.00279701	0.00279605
0.0001699920	0.00281799	0.00293601	0.00293398	0.00281906
0.0021100000	0.00292504	0.00279701	0.00285399	0.00278103
0.0020610100	0.002823	0.00281906	0.023878	
0.0020589800	0.00279796	0.00279593	0.00279701	
0.0021688900	0.002823	0.00281799	0.00292802	
0.0021679400	0.00279891	0.00279605	0.0027951	
0.0020650600	0.00293291	0.00282001	0.00281894	
0.0020550500	0.00279796	0.00279593	0.00279796	
0.0020610100	0.00281799	0.00298989	0.002823	
0.0021659100	0.00281501	0.00279391	0.00279605	
0.0020630400	0.00293195	0.00281107	0.00298607	
0.0021709200	0.00279403	0.00290596	0.00279593	
0.0020589800	0.00282204	0.00281596	0.00282097	
0.0020519500	0.00279903	0.00279605	0.00279295	
0.0020610100	0.00293207	0.00281703	0.00282502	
0.0023119400	0.00279796	0.00279403	0.002859	
0.0020589800	0.00281906	0.00293696	0.00282001	
0.0021649600	0.00285602	0.00279808	0.00279593	
0.00212598	0.00282395	0.002823	0.00294006	
0.00205505	0.00279701	0.00279403	0.00279701	
0.00205898	0.002823	0.00281692	0.00297093	
0.00205505	0.00280094	0.00279498	0.00279403	
0.00206196	0.00299001	0.00288188	0.00282001	
0.00205195	0.00285995	0.00279701	0.00279403	
0.00206304	0.00281703	0.00301504	0.00282001	
0.00223601	0.00290501	0.00280106	0.00290489	
0.000611067	0.00281799	0.00288308	0.00293601	
0.000177979	0.00279999	0.00279903	0.00279701	

Read Local File Between Call Interval Descriptive Statistics

The Data object has **124** elements in its array

DESCRIPTIVE STATISTICS

Minimum = 1.69992E-4
Maximum = 0.023878
Range = 0.023708008
Sum = 0.3536384799999999
Mean = 0.002851923225806451
median = 0.00279897
Variance = 4.50934949772738E-6
Std Dev = 0.0021235228978580332
Skewness = 8.336133910535583
Kurtosis = 80.37299019256903
95% CI = 3.7528435864183767E-4 conducted with sample StdDev NOT population StdDev

Read Local File System Call Interval Data

4.2804700	4.3248200	5.0572000	4.2871400	4.2779500
4.2841100	4.3633400	4.8517200	4.2793300	4.2727300
4.2767800	4.3813400	4.2897700	4.2685100	4.2752200
4.2883300	4.8514900	4.2781600	4.2905900	4.3051400
4.4577200	4.3358400	4.2650800	4.3221000	
4.2826300	4.2886700	4.2838700	4.2846100	
4.2765400	4.2821100	4.2656600	4.2981500	
4.2876900	4.2636600	4.2878800	4.3124700	
4.3591400	4.3370800	4.2700800	4.2706600	
4.2689400	4.2719700	4.3264600	4.2770500	
5.2796300	4.2914000	4.2860700	4.3732800	
5.8598100	4.2747400	4.2713500	4.2744500	
4.9625300	4.2895200	4.4054700	4.3359100	
5.1148200	4.2917600	4.2724700	4.2859500	
5.5357400	4.2637300	4.2857800	4.2825800	
4.2941700	4.3723800	4.2873200	4.2754500	
4.3399600	4.3196500	4.2764100	4.2761000	
4.2847000	4.2878600	4.3370800	4.2735200	
4.2887200	4.2811500	4.2635900	4.2808300	
4.4864200	4.2884400	4.2868200	4.2807100	
4.3925000	4.2715200	4.2872100	4.3273500	
4.2788300	4.2858900	4.3940100	4.2799300	
4.2915100	4.2824600	4.2889800	4.2797300	
4.2805600	4.3515000	4.2904000	4.2771400	
4.2733500	4.3490200	4.2801300	4.3448400	
4.3404800	4.2730500	4.2690900	4.2845000	
4.2778900	4.2885100	4.4165500	4.2836200	
4.2937900	4.6632300	4.2707600	4.2767000	
4.2794100	5.1722800	4.2835200	4.3276600	
4.2838300	5.6836200	4.2850900	4.2789000	

Read Local File System Call Interval Descriptive Statistics

The Data object has **124** elements in its array

DESCRIPTIVE STATISTICS

Minimum = 4.26359
Maximum = 5.85981
Range = 1.5962200000000006
Sum = 542.9178100000001
Mean = 4.378369435483872
median = 4.286445
Variance = 0.07669490164276818
Std Dev = 0.27693844377906107
Skewness = 3.6271420741918203
Kurtosis = 13.337311135732914
95% CI = 0.04894256914381611 conducted with sample StdDev NOT population StdDev

Write Local File Between Call Interval Data

0.0156031	0.0047240	0.0048000	0.0017691	0.0038041
0.0073600	0.0047650	0.0050629	0.0034690	
0.0084820	0.0047050	0.0048571	0.0017691	
0.0057191	0.0047389	0.0048400	0.0025539	
0.0088260	0.0053140	0.0050490	0.0009520	
0.0058780	0.0060949	0.0048760	0.0010110	
0.0082260	0.0049460	0.0048801	0.0015650	
0.0055970	0.0051309	0.0048140	0.0009500	
0.0079370	0.0054439	0.0049889	0.0009520	
0.0048281	0.0044271	0.0051869	0.0009490	
0.0071050	0.0456740	0.0052869	0.0009480	
0.0063371	0.0830749	0.0053390	0.0009490	
0.0079560	0.0780929	0.0049470	0.0009520	
0.0047760	3.0012600	0.0050730	0.0009481	
0.0066360	0.0065221	0.0049139	0.0009500	
0.0032510	0.0065230	0.0049650	0.0009509	
0.0077579	0.0064651	0.0047780	0.0009511	
0.0036781	0.0057050	0.0049460	0.0009490	
0.0080320	0.0056670	0.0047720	0.0009470	
0.0028909	0.0056560	0.0048940	0.0009550	
0.0090940	0.0060140	0.0048590	0.0009500	
0.0033259	0.0051100	0.0049939	0.0009511	
0.0098050	0.0048591	0.0048341	0.0009480	
0.0027220	0.0049671	0.0060340	0.0009490	
0.0080670	0.0047621	0.0048890	0.0009569	
0.0002149	0.0048649	0.0046971	0.0009520	
0.0016660	0.0047571	0.0047809	0.0009481	
0.0090051	0.0048111	0.0046860	0.0009511	
0.0058440	0.0047209	0.0047491	0.0009490	
0.0056130	0.0048590	0.0046630	0.0005450	

Write Local File Between Call Interval

Descriptive Statistics

The Data object has **121** elements in its array

DESCRIPTIVE STATISTICS

Minimum = 2.14934E-4
Maximum = 3.00126
Range = 3.0010450659999997
Sum = 3.723687764
Mean = 0.030774279041322315
median = 0.00484002
Variance = 0.07425892790543169
Std Dev = 0.27250491354364914
Skewness = 10.974487295973677
Kurtosis = 120.6173104054261
95% CI = 0.04875731547175822 conducted with sample StdDev NOT population StdDev

Write Local File System Call Interval Data

0.0005690	0.0005190	0.0005051	0.0006729	0.0005070
0.0005760	0.0004480	0.0004460	0.0004431	
0.0006280	0.0005220	0.0005920	0.0005330	
0.0005569	0.0004520	0.0004300	0.0004431	
0.0006289	0.0005701	0.0005740	0.0004820	
0.0005690	0.0004400	0.0004330	0.0005420	
0.0006280	0.0005771	0.0005579	0.0004890	
0.0005630	0.0004561	0.0004281	0.0005630	
0.0006330	0.0005640	0.0005341	0.0004901	
0.0005519	0.0004290	0.0004281	0.0006260	
0.0006210	0.0006330	0.0005630	0.0004860	
0.0005690	0.0005311	0.0004910	0.0005389	
0.0006419	0.0006311	0.0005320	0.0004910	
0.0005510	0.0005779	0.0004431	0.0005610	
0.0006059	0.0005629	0.0005350	0.0004921	
0.0005391	0.0004500	0.0004461	0.0005660	
0.0006311	0.0005640	0.0005410	0.0004940	
0.0006330	0.0004380	0.0004420	0.0005660	
0.0006050	0.0005680	0.0005360	0.0004849	
0.0005460	0.0004380	0.0004491	0.0005270	
0.0006270	0.0005651	0.0005420	0.0004880	
0.0005521	0.0004629	0.0004660	0.0005530	
0.0006360	0.0005499	0.0005410	0.0004920	
0.0005389	0.0004519	0.0004810	0.0005610	
0.0006450	0.0005450	0.0005829	0.0004860	
0.0004971	0.0004480	0.0004320	0.0005580	
0.0004960	0.0005460	0.0005530	0.0004870	
0.0017310	0.0004860	0.0004300	0.0005670	
0.0005690	0.0005420	0.0005370	0.0004820	
0.0004770	0.0004450	0.0004690	0.0005630	

Write Local File System Call Interval Descriptive Statistics

The Data object has **121** elements in its array

DESCRIPTIVE STATISTICS

Minimum = 4.28081E-4
Maximum = 0.00173104
Range = 0.001302959
Sum = 0.06520473700000001
Mean = 5.388821239669423E-4
median = 5.38945E-4
Variance = 1.5787897999059505E-8
Std Dev = 1.2564990250318344E-4
Skewness = 7.188071883660956
Kurtosis = 68.05663344645232
95% CI = 2.2481620076777346E-5 conducted with sample StdDev NOT population StdDev

Write Remote File Between Call Interval Data

0.0001320	0.0001349	0.0001349	0.0001330	0.0001360
0.0001321	0.0001309	0.0001370	0.0001351	0.0001320
0.0001320	0.0001320	0.0001340	0.0001310	0.0001321
0.0001329	0.0001340	0.0001321	0.0001320	0.0001301
0.0001330	0.0001320	0.0001329	0.0001400	0.0001340
0.0001321	0.0001320	0.0001310	0.0001370	0.0001330
0.0001310	0.0001340	0.0001321	0.0001321	0.0001340
0.0001330	0.0001340	0.0001330	0.0001961	
0.0001329	0.0001310	0.0001329	0.0001329	
0.0001409	0.0001330	0.0001330	0.0001340	

Write Remote File Between Call Interval Descriptive Statistics

The Data object has 47 elements in its array

DESCRIPTIVE STATISTICS

Minimum = 1.30057E-4

Maximum = 1.96099E-4

Range = 6.604200000000002E-5

Sum = 0.006326914999999999

Mean = 1.3461521276595742E-4

median = 1.32918E-4

Variance = 8.848993495374656E-11

Std Dev = 9.406908894729796E-6

Skewness = 6.334653681690928

Kurtosis = 42.011280023941055

95% CI = 2.7184671756673306E-6 conducted with sample StdDev NOT population StdDev

Write Remote File System Call Interval Data

0.0007430	0.0007371	0.0007130	0.0007449	0.0023741
0.0007360	0.0031611	0.0007250	0.0025040	0.0007470
0.0007380	0.0007499	0.0007401	0.0007390	0.0007399
0.0023891	0.0007451	0.0023190	0.0007430	0.0032979
0.0007380	0.0007510	0.0007430	0.0007560	0.0007440
0.0007130	0.0024420	0.0007451	0.0031530	0.0007470
0.0007490	0.0007370	0.0007410	0.0007380	0.0007499
0.0025049	0.0007360	0.0032489	0.0007509	
0.0007340	0.0007460	0.0007380	0.0007421	
0.0007441	0.0028380	0.0007449	0.0023710	

Write Remote File System Call Interval Descriptive Statistics

The Data object has 47 elements in its array

DESCRIPTIVE STATISTICS

Minimum = 7.12991E-4

Maximum = 0.00329792

Range = 0.002584929

Sum = 0.058531985999999994

Mean = 0.001245361404255319

median = 7.44939E-4

Variance = 7.954153236570723E-7

Std Dev = 8.918605965379748E-4

Skewness = 1.328469203612622

Kurtosis = 0.028706766384108295

95% CI = 2.5773543510322367E-4 conducted with sample StdDev NOT population StdDev

APPENDIX C: EXPERIMENT RESULTS

Read Local File Between Call Interval Results

TEST: K-S Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted K-S test statistic 5.017385802574527 is greater than the modified critical value 1.035. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: K-S Test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted K-S test statistic 5.389212805739073 is greater than the modified critical value 1.308. Therefore, reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

TEST: A-D Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted A-D test statistic 30.677626699552437 is greater than the modified critical value 1.029. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: A-D Test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted A-D test statistic 35.577424141814724 is greater than the modified critical value 1.943. Therefore reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

Read Local File System Call Interval Results

TEST: K-S Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted K-S test statistic 4.022949627920754 is greater than the modified critical value 1.035. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: K-S Test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted K-S test statistic 7.101488327130617 is greater than the modified critical value 1.308. Therefore, reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

TEST: A-D Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted A-D test statistic 29.938894071092253 is greater than the modified critical value 1.029. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: A-D Test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted A-D test statistic 53.527837656652814 is greater than the modified critical value 1.943. Therefore reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

Write Local File Between Call Interval Results

TEST: K-S Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted K-S test statistic 5.38801483546206 is greater than the modified critical value 1.035. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: K-S Test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted K-S test statistic 7.833717859887696 is greater than the modified critical value 1.308. Therefore, reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

TEST: A-D Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted A-D test statistic 45.53619992778167 is greater than the modified critical value 1.029. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: A-D Test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted A-D test statistic 104.13662957631682 is greater than the modified critical value 1.943. Therefore reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

Write Local File System Call Interval Results

TEST: K-S Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted K-S test statistic 2.357447995492111 is greater than the modified critical value 1.035. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: K-S Test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted K-S test statistic 6.178306629193417 is greater than the modified critical value 1.308. Therefore, reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

TEST: A-D Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted A-D test statistic 10.160396396726927 is greater than the modified critical value 1.029. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: A-D Test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted A-D test statistic 42.19919343201918 is greater than the modified critical value 1.943. Therefore reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

Write Remote File Between Call Interval Results

TEST: K-S Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted K-S test statistic 2.48335644275209026 is greater than the modified critical value 1.035. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: K-S Test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted K-S test statistic 4.422383945811191 is greater than the modified critical value 1.308. Therefore, reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

TEST: A-D Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted A-D test statistic 11.454589644697467 is greater than the modified critical value 1.029. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: A-D Test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted A-D test statistic 20.68485876216289 is greater than the modified critical value 1.943. Therefore reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

Write Remote File System Call Interval Results

TEST: K-S Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted K-S test statistic 3.1369014322242457 is greater than the modified critical value 1.035. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: K-S Test for exponential distribution

NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted K-S test statistic 3.1028894423071307 is greater than the modified critical value 1.308. Therefore, reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

TEST: A-D Test for normal distribution

NULL HYPOTHESIS: Data is from a population with an underlying normal distribution

RESULTS: The adjusted A-D test statistic 10.11692228867237 is greater than the modified critical value 1.029. Therefore, reject the null hypothesis that the data is from a normal distribution using a 99% CI.

TEST: A-D Test for exponential distribution

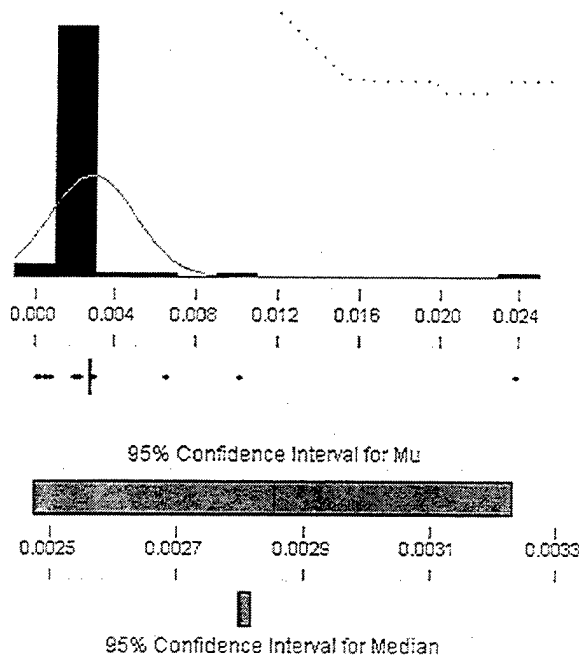
NULL HYPOTHESIS: Data is from a population with an underlying exponential distribution

RESULTS: The adjusted A-D test statistic 8.704621245573474 is greater than the modified critical value 1.943. Therefore reject the null hypothesis that the data is from an exponential distribution using a 99% CI.

Appendix D: minitab results

Descriptive Statistics

Variable: rlfbcj



Anderson-Darling Normality Test

A-Squared: 29.469
P-Value: 0.000

Mean 2.85E-03
StDev 2.12E-03
Variance 4.51E-06
Skewness 8.33614
Kurtosis 80.3731
N 124

Minimum 1.70E-04
1st Quartile 2.78E-03
Median 2.80E-03
3rd Quartile 2.82E-03
Maximum 2.39E-02

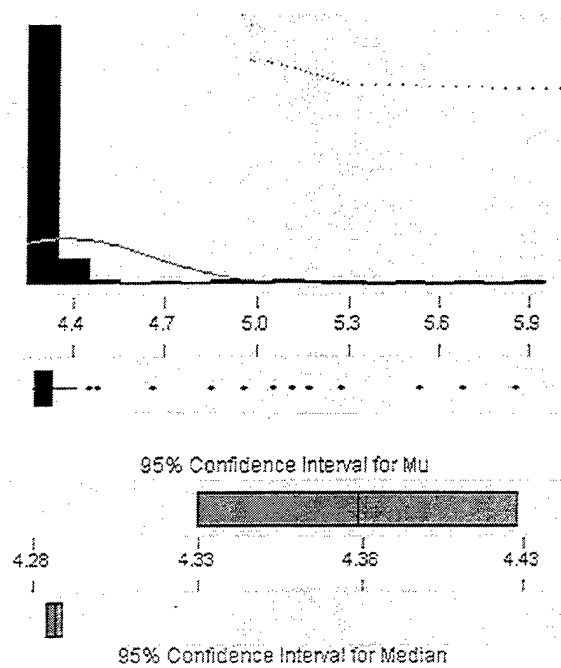
95% Confidence Interval for Mu
2.47E-03 3.23E-03

95% Confidence Interval for Sigma
1.89E-03 2.43E-03

95% Confidence Interval for Median
2.80E-03 2.82E-03

Figure 21: Minitab Results For rlfbcj Data

Descriptive Statistics



Variable: rlfsci

Anderson-Darling Normality Test

A-Squared: 29.406
P-Value: 0.000

Mean 4.37837
StDev 0.27694
Variance 7.67E-02
Skewness 3.62714
Kurtosis 13.3373
N 124

Minimum 4.26359
1st Quartile 4.27791
Median 4.28645
3rd Quartile 4.33380
Maximum 5.85981

95% Confidence Interval for Mu

4.32914 4.42760

95% Confidence Interval for Sigma

0.24623 0.31646

95% Confidence Interval for Median

4.28375 4.28869

Figure 22: Minitab Results For rlfsci Data

Descriptive Statistics

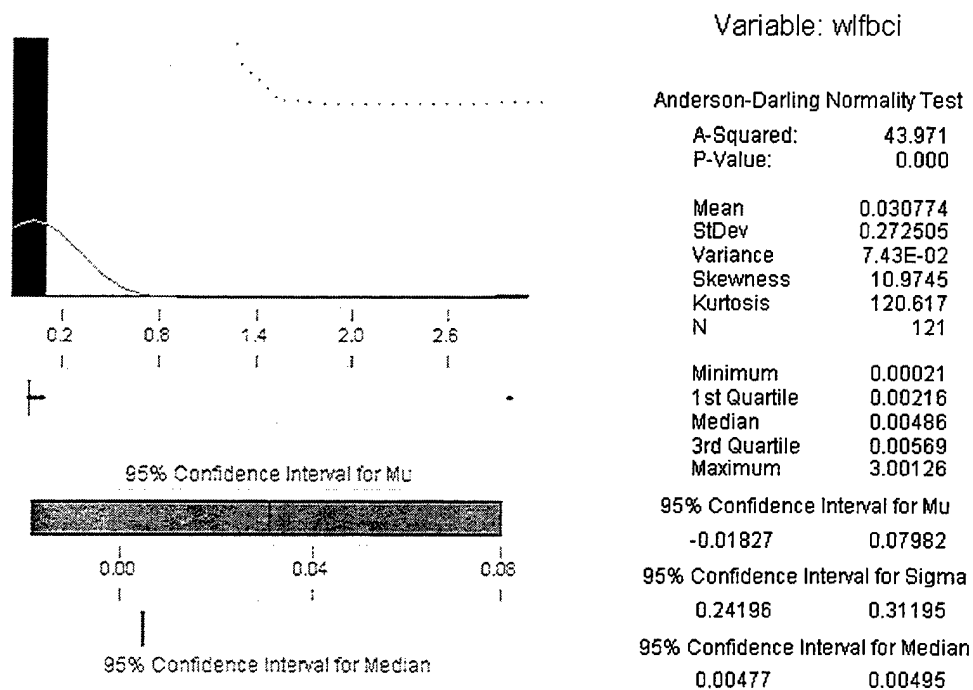
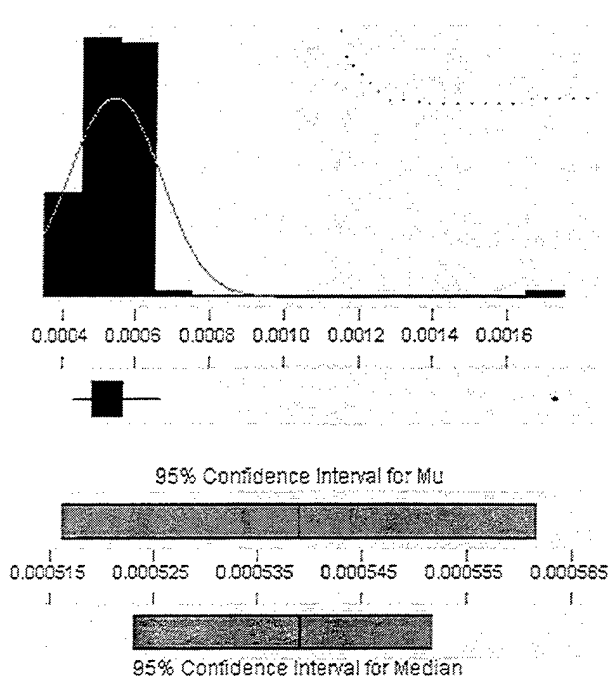


Figure 23: Minitab Results For rlfsci Data

Descriptive Statistics



Variable: wlfsci

Anderson-Darling Normality Test

A-Squared: 9.594
P-Value: 0.000

Mean 5.39E-04
StDev 1.26E-04
Variance 1.58E-08
Skewness 7.18826
Kurtosis 68.0590
N 121

Minimum 4.28E-04
1st Quartile 4.82E-04
Median 5.39E-04
3rd Quartile 5.68E-04
Maximum 1.73E-03

95% Confidence Interval for Mu

5.16E-04 5.61E-04

95% Confidence Interval for Sigma

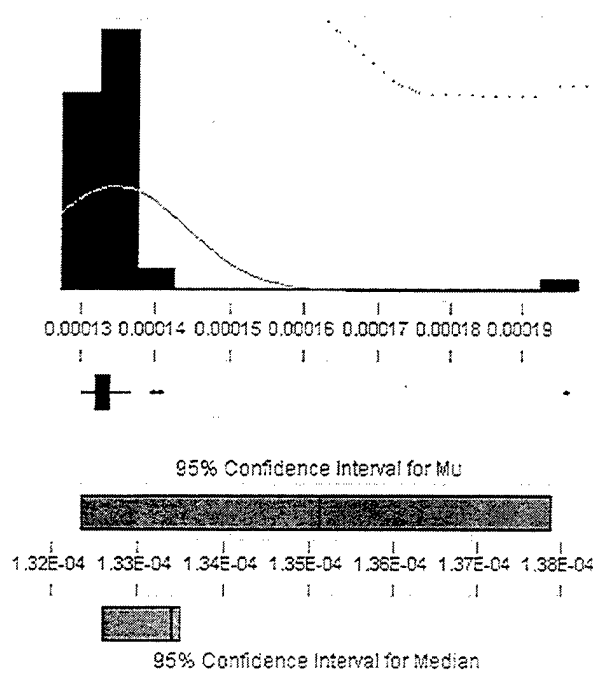
1.12E-04 1.44E-04

95% Confidence Interval for Median

5.23E-04 5.52E-04

Figure 24: Minitab Results For wlfsci Data

Descriptive Statistics



Variable: wrfbci

Anderson-Darling Normality Test

A-Squared: 11.011
P-Value: 0.000

Mean 1.35E-04
StDev 9.41E-06
Variance 8.85E-11
Skewness 6.33475
Kurtosis 42.0100
N 47

Minimum 1.30E-04
1st Quartile 1.32E-04
Median 1.33E-04
3rd Quartile 1.34E-04
Maximum 1.36E-04

95% Confidence Interval for Mu
1.32E-04 1.37E-04

95% Confidence Interval for Sigma
7.82E-06 1.18E-05

95% Confidence Interval for Median
1.32E-04 1.33E-04

Figure 25: Minitab Results For wrfbci Data

Descriptive Statistics

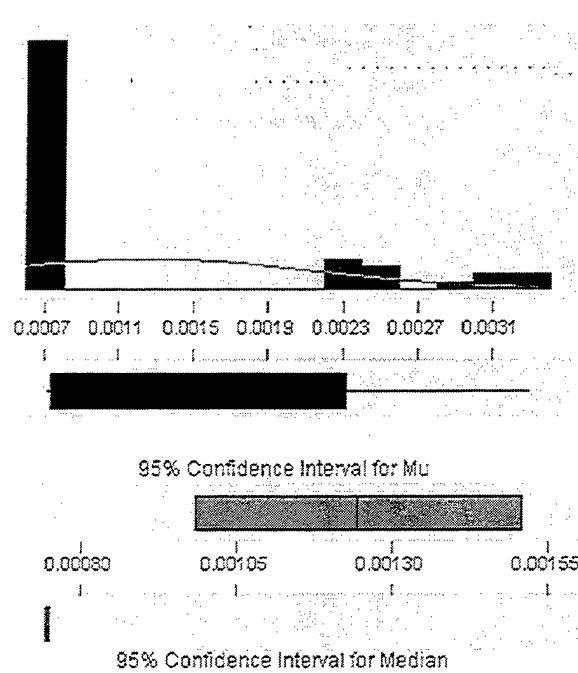


Figure 26: Minitab Results For wrfsci Data

APPENDIX E: ACRONYMS

cdf	Cumulative distribution function
CL	Client Library
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DoD	Department of Defense
ETC	Expected Time for Completion
fd	File descriptor
I/O	Input and/or Output
MSHN	Management System for Heterogeneous Networks
NWS	Network Weather Service
OLB	Opportunistic Load Balancing
OS	Operating System
QoS	Quality of Service
RDT&E	Research, Development, Testing, and Evaluation
rlfbcf	read local file between call interval
rlfsci	read local file system call interval
RMS	Resource Management System
RRD	Resource Requirements Database
RSS	Resource Status Server
SA	Scheduling Advisor
wlfbcf	write local file between call interval
wlfsci	write local file system call interval
wrfbcf	write remote file between call interval
wrfsci	write remote file system call interval

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [BECK00] K.J. Becker, D.P. Gaver, K.D. Glazebrook, P.A. Jacobs and S. Lawphongpanich, Allocation Of Tasks To Specialized Processors: A planning Approach, To appear European Journal Of Operational Research 2000.
- [BRES96] John L. Bresina, Heuristic-Biased Stochastic Sampling, Proceedings of AAAI, Portland, OR, 1996
- [DESI] DeSiDeRaTa Team, *Resource and QoS Management for Dynamic, Scalable, Dependable Real-Time Systems*, Manual for use of distributed QoS and resource management middleware, Lab for Parallel and Distributed Real-Time Systems, Department of Computer Science and Engineering, University of Texas at Arlington, circa 1998.
- [HENS97] Debbie Hensgen and John Falby, *Draft Proposal for Institute for Joint Warfare Analysis Research*, Monterey, CA, 1997.
- [HENS99] Debbie Hensgen, Taylor Kidd, David St. John, Matthew C. Schnaidt, Howard Jay Siegel, Tracy D. Braun, Muthucumaru Mayheswaran, Shoukat Ali, Jong-Kook Kim, Cynthia Irvine, Tim Levin, Richard F. Freund, Matt Kussow, Michael Godfrey, Alpay Duman, Paul Carff, Shirley Kidd, Viktor Prasanna, Prashanth Bhat, and Ammar Alhusaini, *An Overview of MSHN: The Management System for Heterogeneous Networks*, Proceedings of HCW '99, San Juan, Puerto Rico, 1999.
- [LIUR98] Zhen Liu and Rhonda Righter, Optimal Load Balancing On Distributed Homogeneous Unreliable Processors, Operations Research, Vol. 46, No. 4, July-August 1998
- [KAUF97] Morgan Kaufmann, High Performance Distributed Computing: Building A Computational Grid, Morgan Kaufmann Publishers, 1997.
- [KIDD96] Taylor Kidd, Debbie Hensgen, Richard Freund, and Lantz Moore, "SmartNet: A Scheduling Framework for Heterogeneous Computing," *ISPAN*, 1996.
- [KIDD98] Taylor Kidd, Debra Hensgen, Richard Freund, Matt Kussow, and Mark Campbell. *Compute Characteristics: A Useful Characterization of Job Runtimes*. In preparation for submission (1998).

- [KIDD99] Taylor Kidd and Debra Hensgen. *Why the Mean is Inadequate for Accurate Scheduling Decisions*. In preparation for submission (1999).
- [LAWK91] Averill M. Law and W. David Kelton, *Simulation Modeling & Analysis*, 2nd Edition. McGraw-Hill, Inc. 1991.
- [LIVN95] Miron Livny, Michael Litzkow, Todd Tannenbaum, and Jim Basney, *Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System*, Dr Dobbs Journal, February 1995.
- [NOBL97] Brian Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker, "Agile Application-Aware Adaptation for Mobility," *Proceedings of the 16th Symposium on Operating Systems*, 1997.
- [ROSS91] Keith Ross and David Yao, "Optimal Load Balancing and Scheduling in a Distributed Computer System", *Journal of the Association for Computing Machinery*, Vol. 38, No. 3, July 1991.
- [SCHN99] Matt Schnaidt, Debra Hensgen, John Falby, Taylor Kidd, and David St. John, *Passive, Domain-Independent, End-to-End Message Passing Performance Monitoring to Support Adaptive Applications in MSHN*, submitted to OSDI 99.
- [WOLS97] Rich Wolski, Neil Spring, and Christopher Peterson, *Implementing a Performance Forecasting System for Metacomputing: The Network Weather Service*, SC97 Technical Paper, 1997.